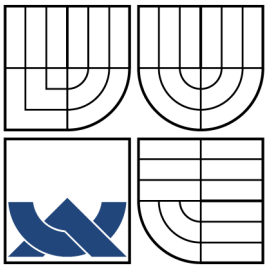
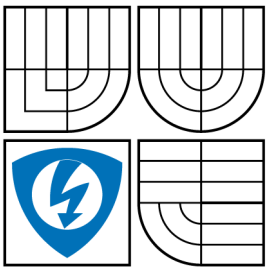


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE DETEKTORŮ HLASOVÉ AKTIVITY S VYUŽITÍM OPEN-SOURCE KNIHOVEN V JAZYCE C

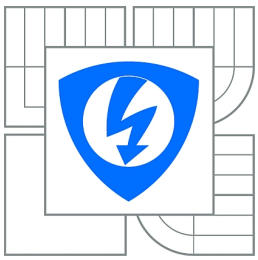
IMPLEMENTATION OF VOICE ACTIVITY DETECTORS USING OPEN-SOURCE LIBRARIES IN C LANGUAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. VÁCLAV MACH

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. IVAN MÍČA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Václav Mach	***ID:***	73003
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Implemetace detektorů hlasové aktivity s využitím open-source knihoven v jazyce C

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a v jazyce C implementujte systém pro detekci hlasové aktivity (VAD) v audio nahrávkách běžných formátů. Systém má sloužit rovněž k orientačnímu vyhodnocení úspěšnosti detekce pomocí informací v souborech značek příslušných k testovaným nahrávkám. Pro implementaci použijte volně dostupných knihoven s otevřeným zdrojovým kódem. Porovnejte rychlosti implementací v C a v Matlabu.

DOPORUČENÁ LITERATURA:

[1] PSUTKA, J., et al. Mluvíme s počítačem česky. Praha: Academia, 2006. 752 s. ISBN 80-200-1209-1.
[2] KACUR, J. The Concept of Task Specific Speech Database for VAD Systems. In 48th International Symposium ELMAR-2006., s. 155-158. ISBN 953-7044-03-3.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Ivan Míča

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato diplomová práce pojednává o problematice detekce hlasové aktivity. Jsou zde teoreticky popsány dva typy: energetický a statistický detektor, jejichž funkce je ověřena v prostředí Matlab. Dále je řešena implementace detektorů v jazyce C za pomoci standardních knihoven a open-source knihoven GSL. Realizované algoritmy jsou porovnány z hlediska rychlosti výpočtu, paměťové náročnosti a zátěže jednotlivých matematických operací v rámci celého algoritmu. Bylo provedeno porovnání rychlosti výpočtu v závislosti na velikosti zpracovávaného okna.

KLÍČOVÁ SLOVA

detekce, řečové, aktivity, VAD, statistický, energetický, valgrind, oprofile, procesorový, čas

ABSTRACT

This diploma thesis discusses the issue of Voice Activity Detection. There are two types of detectors described: energetic and statistics. Their funktionality is proved in the Matlab environment. Further, the implemetation of VADs is made through C language with standard libraries and GSL open-source libraries. The realized algorithms are compared in the scope of processing time of computation, memory management and a single mathematical operations stress. Also a comparism of the processing time according to segment length was made.

KEYWORDS

voice, activity, detector, VAD, statistical, energy, valgrind, oprofile, processing, time

MACH, Václav *Implementace detektorů hlasové aktivity s využitím open-source knihoven v jazyce C*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 60 s. Vedoucí práce byl Ing. Ivan MÍČA

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Implementace detektorů hlasové aktivity s využitím open-source knihoven v jazyce C“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Poděkování

Mé poděkování patří v první řadě mému vedoucímu práce Ing. Ivanu Míčovi za cenné rady a vedení při realizaci této diplomové práce. Poděkování patří taktéž celé mé rodině a hlavně rodičům za podporu a toleranci při dokončování práce.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Detekce řečové aktivity je součástí mnoha různých systémů pro zpracování řečových signálů. Algoritmus řešící tuto úlohu se nazývá detektor hlasové aktivity (angl. Voice Activity Detector, zkráceně VAD). Ve většině případů jsou aplikovány ve větších komplexních systémech, např. pro rozpoznávání řeči, přenos a kompresi řeči, zvýrazňování řeči. Podle příznaků, které jsou pro detekci řeči brány v potaz, se rozlišují jednotlivé typy detektorů. Příkladem rozhodovacích faktorů můžou být energie, sledování průchodů nulou, analýza kepstrálních koeficientů, periodicita signálu nebo rozložení koeficientů diskrétní Fourierovy řady (DFŘ).

Lidský hlas představuje širokou škálu různých typů signálu v časové i frekvenční doméně, čímž dostáváme různé možnosti interpretace řečového signálu. Významným fenoménem je vždy přítomný hluk na pozadí, taktéž různě interpretovatelný v libovolné doméně. Toto jsou hlavní důvody proč nemůžeme implementovat ideální algoritmus VAD. Proto se musí detektor řešit individuálně s ohledem na konkrétní aplikaci systému.

Úkolem této diplomové práce je implementace detektorů hlasové aktivity v jazyce C za použití volně dostupných open-source knihoven. Dále je popsáno několik způsobů měření procesorového času a na jejich základě jsou jednotlivé implementace porovnány s hlediska doby výpočtu. Ta je porovnána s rychlostí výpočtu v prostředí Octave.

Taktéž je srovnána paměťová náročnost algoritmů detektorů hlasové aktivity a zaměřujeme se i na konkrétní náročnost výpočtu jednotlivých funkcí v rámci celého algoritmu. Oba algoritmy jsou porovnány se zaměřením na dobu výpočtu detekce hlasové aktivity v závislosti na zvolené délce segmentu.

K účelu testování je vytvořena minimální databáze označkovaných nahrávek vět typických pro prostředí s daným hlukem na pozadí o různých poměrech odstupu signálu od hluku (Signal to Noise Ratio, zkr. SNR).

1 DETEKTOR ŘEČOVÉ AKTIVITY

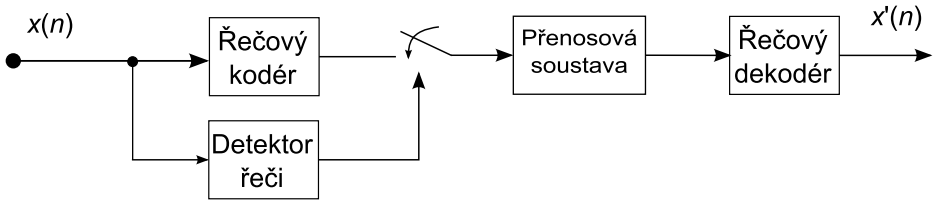
1.1 Základní funkce detektoru

Základní funkcí všech algoritmů VAD je získání nějaké měřené veličiny nebo množství ze vstupního signálu a srovnání těchto hodnot s prahy obvykle získaných z charakteristik hluku na pozadí a hlasového signálu. VAD v nestacionárním hluku potřebuje časově variantní prahové hodnoty, které se počítají v hlasově neaktivních segmentech. Protože VAD algoritmy bývají součástí aplikací pro zpracování řeči, je rozhodujícím článkem pro jejich výkon implementovaný VAD [19].

Výpočtu hledaných charakteristik musí předcházet několik kroků, které jsou pro všechny algoritmy společné. Prvním je rozdělení vstupního signálu do segmentů s daným přesahem. V systémech, kde není vyžadováno zpracování signálu v reálném čase může být délka segmentu libovolně dlouhá. Volba délky segmentu závisí pouze na schopnosti objektivního ohodnocení řečového/neřečového signálu. V systémech se zpracováním řeči v reálném čase je však nutné dbát ohled na rychlost výpočtu vzhledem k velikosti segmentu.

Dalším krokem je zpracování daného segmentu, kde výsledkem je vektor vybraných měřených nebo spočítaných parametrů. Následně přichází porovnání dvou segmentů a určení jejich odstupu. Toto ohodnocení může probíhat dvěma způsoby. Pokud je porovnáván aktuálně zpracovávaný segment s předem nastaveným prahem, potom se jedná o integrační algoritmus, který je schopen detekovat celý interval řečové aktivity. Požadavkem pro tento systém je subjektivní nastavení prahové hodnoty.

V případě, že porovnáváme dva sousední segmenty, potom mluvíme o diferenčním algoritmu, jehož výsledkem může být pouze detekce hranice řečového úseku. Nastavení předem zvoleného prahu zde není nutné, ovšem pokud chceme, aby výsledkem algoritmu bylo označení celého intervalu řečové aktivity, je nutné ještě následné zpracování, kterým rozumíme integraci. Toto postzpracování způsobuje často chyby a degraduje tak výkon diferenčních algoritmů [16].



Obr. 1.1: Příklad zapojení systému detekce řečové aktivity v komunikačním systému.

1.2 Faktory ovlivňující přesnost detekce

V komunikačních systémech se většinou sleduje průběh řečové aktivity na vstupu z mikrofonu nepřetržitě a jeho příznaková reprezentace je následně předávána kodéru. Přesná detekce počátku a konce řečové aktivity je nutná zejména při rozpoznávání izolovaných slov a krátkých frází. Při nepřesné nebo pozdní detekci může mít toto zpoždění zásadní vliv na srozumitelnost přenášené informace. Při rozpoznávání delších úseků řeči musí být přesnost klasifikace nastavena tak, aby byl systém schopen zachytit hlavně začátky řečové aktivity a při rozpoznávání jednotlivých slov neuzavírat zbytečně kanál, aby byla zachována srozumitelnost.

Pokud má signál dostatečný odstup od hluku na pozadí a artikulace fráze je srozumitelná, není problém detekovat počátky i konce promluvy, ale i souhlásky s nízkou energií, např. neznělé sykavky, protože převyšují úroveň okolního šumu. V reálných případech však těchto ideálních podmínek nemůžeme dosáhnout, protože rušení často dosahuje vysoké úrovně.

Jevy, které mají na rušení řečového signálu podstatný vliv se dají rozdělit do tří hlavních skupin [11, str. 172-174]:

1. **Akustické pozadí** řeči bývá rušivé zejména pokud do užitečného signálu řeči zasahuje např. vzdálený hovor, motor vozidla, bouchání dveří atd. Problémem bývají zejména hlasité periodické signály.
2. **Rušení**, jako např. tónové interference nebo intermodulační zkreslení, které produkuje přenosový kanál. K nim je nutné přistupovat s ohledem pokud přenášíme signál např. telefonní linkou.
3. **Nedokonalá artikulace** degraduje kvalitu řeči, pokud mluvčí během promluvy např. mlaská nebo hlasitě dýchá. Tyhle zvuky můžou taktéž dosáhnout energetické úrovně užitečného signálu.

Tyto faktory nabývají významu hlavně když se na začátku nebo konci objevují jevy jako:

- neznělé okluzivy, tj. „p“, „t“, „k“
- znělé okluzivy, tj. „b“, „d“, „g“ na konci promluvy, kdy se mění na neznělé protějšky „p“, „t“, „k“;
- slabé neznělé frikativy, např. „f“, „s“, „š“, „ch“;
- znělé frikativy „v“, „z“, „ž“, „h“ na konci promluvy, kdy se mění na neznělé protějšky „f“, „s“, „š“, „ch“;
- nosní souhlásky „m“, „n“, „ň“ na konci promluvy.

1.3 Odhad neznámých parametrů

V úloze statistického detektoru hlasové aktivity se potkáváme s problémem, kde na základě naměřených hodnot náhodné proměnné musíme co nejlépe odhadnout její rozdělení a nebo parametry jejího rozdělení. Protože hodnoty neznámých parametrů nebo rozdělení pravděpodobnosti odhadujeme na základě pozorovaných hodnot náhodné proměnné, nikdy nemůžeme dostat přesné výsledky těchto hodnot. Z toho důvodu nazýváme tuto úlohu odhadem parametrů nebo konkrétně v našem případě odhad rozdělení pravděpodobnosti. Na základě předpokladu, že statistický detektor řečové aktivity je založen na základě normálního (Gaussovského) rozdělení, bude dále použit odhad neznámých parametrů tohoto rozdělení náhodné veličiny.

1.3.1 Náhodný výběr

Vlastnosti statistických znaků (náhodných proměnných) jsou popsány jejich zákonem rozdělení. Uvažujme experiment, jehož výsledkem je nějaká hodnota x statistického znaku. To odpovídá tvrzení, že tento výsledek je náhodná proměnná s rozdělením $f(x, \theta)$, kde θ je neznámý parametr. Experiment nezávisle opakujeme n-krát a dostaneme tak posloupnost n nezávislých měření $x_1, x_2, x_3, \ldots, x_n$. Uvedenou posloupnost můžeme považovat za realizaci nezávislých náhodných proměnných $X_1, X_2, X_3, \ldots, X_n$ s tím samým zákonem rozdělení $f(x, \theta)$. Tato posloupnost se nazývá náhodný výběr, kde θ je reálný parametr a n udává rozsah náhodného výběru.

Množinu všech možných realizací náhodného výběru $(X_1, X_2, X_3, \ldots, X_n)'$ nazýváme výběrovým prostorem. Množina hodnot, které může parametr θ nabývat se nazývá parametrickým prostorem a je označován Θ. Pokud uvažujeme rozdělení $N(\mu, \sigma^2)$, kde μ, σ^2 jsou neznámé parametry, pak parametrickým prostorem Θ je polorovina $-\infty < \mu < \infty, \sigma^2 > 0$ [9, str. 70-71].

1.3.2 Metoda bodového odhadu

Informace o neznámém parametru získáme na základě náhodného výběru, který má konečný rozsah a parametry je možno popsat charakteristikami uvedenými v předchozí kapitole. Dále je hledána měřitelná funkce výběrových hodnot, které budou co nejblíže k hodnotě odhadovaného parametru. Každý odhad je měřitelnou funkcí složek náhodného výběru a proto je taky náhodnou proměnnou.

Posloupnost navzájem nezávislých náhodných proměnných $X_1, X_2, X_3, \ldots, X_n$ nazveme náhodným výběrem z rozdělení $f(x, \theta)$, kde θ je reálný parametr. Jejich realizace je označována $x_1, x_2, x_3, \ldots, x_n$. Nechť T je měřitelné zobrazení z $(R^n, \mathcal{B}_n)$ do $(R, \mathcal{B})$, potom každou funkci $T_n = T(X_1, \ldots, X_n)$, pomocí které odhadujeme neznámý parametr θ budeme nazývat výběrový odhad parametru θ. Odhad T_n závisí

pouze na X_i, $i = 1, 2, \ldots, n$ a nezávisí na parametru θ. Kvalita odhadu závisí jak na vhodné funkci, tak na rozsahu výběru a jako lepší odhad je považován ten, kteréhož rozdělení je více koncentrované okolo neznámé hodnoty parametru. Tento požadavek vyjadřujeme pomocí střední hodnoty a rozptylu.

Odhad T_n, pro který platí

$$\mathrm{E}(T_n) = 0 \text{ pro každé } \theta \in \Theta, \tag{1.1}$$

nazýváme nevychýleným odhadem parametru θ. Pokud platí nerovnost $\mathrm{E}(T_n) > 0$ pro každé $\theta \in \Theta$, nazýváme takový odhad kladně vychýlený, v opačném případě záporně vychýlený. Nutno však upozornit, že vlastnost nevychýlenosti ještě neposkytuje záruku dobrého odhadu, pouze vylučuje systematické chyby.

Pokud je zvětšován rozsah výběru, měl by být zaručen neomezený růst pravděpodobnosti k jistotě, že odhad T_n se nevzdálí od skutečné hodnoty parametru o víc, jako o libovolně malé kladné číslo ϵ. Pokud odhad T_n konverguje podle pravděpodobnosti k parametru θ, nazýváme jej konzistentním odhadem parametru θ a pro libovolné $\epsilon > 0$ platí

$$\lim_{n\to\infty} P\{|\, T_n - \theta| \geq \epsilon\} = 0. \tag{1.2}$$

Používání konzistentních odhadů zaručuje malou pravděpodobnost velké chyby v odhadu parametru, pokud rozsah výběru dostatečně roste. Konzistenci odhadu lze ověřit podle [9, str. 81].

Pokud máme více odhadů T parametru θ, které jsou nevychýlené, určíme nejlepší odhad tak, že hledáme nejmenší disperzi mezi všemi nevychýlenými odhady. Nejlepší odhad nazýváme efektivním odhadem. Jeho nalezení však není jednoduché, proto musíme redukovat výběr, ze kterého vybíráme. Toho dosáhneme pomocí dostačujících statistik.

Nechť $\mathbf{X} = (X_1, X_2, X_3, \ldots, X_n)'$ je náhodný výběr. Statistiku $T(\mathbf{X})$ nazýváme postačující statistikou pro parametr θ patřící do parametrického prostoru $\theta \in \Theta$ pokud sdružené rozdělení náhodného výběru $\mathbf{X} = (X_1, X_2, X_3, \ldots, X_n)'$ podmíněné jevem $T(\mathbf{X}) = t$ je pro každé t nezávislé na parametru $\theta \in \Theta$. Tato postačující statistika poskytuje celou informaci o hodnotě parametru θ, kterou výsledek experimentu obsahuje. Urychlení hledání postačujících statistik a rozhodnutí, zda je statistika postačující definuje věta o faktorizaci uvedená v [9, str. 83].

Metoda momentů a metoda maximální věrohodnosti patří k nejčastěji používaným metodám pro hledání bodového odhadu. Metoda momentů je starší a jednoduchá metoda, která se používá především tehdy, pokud jsou další metody numericky náročnější. Není ani příliš přesná, proto se používá pouze k počátečním aproximacím. V dalším textu bude popsána metoda maximální věrohodnosti, která podává lepší výsledky a její vlastnosti jsou v určitém smyslu asymptoticky optimální [9, str. 79-84].

1.3.3 Metoda maximální věrohodnosti

Předpokládáme, že náhodný výběr $X_1, X_2, X_3, \ldots, X_n$ je z jednorozměrného rozdělení $f(x, \theta)$, závislého na jednom reálném parametru. Tento náhodný výběr je označen $\mathbf{X}$, a dále $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)'$ je jeho realizace. Rozdělení, ze kterého byl proveden výběr, má hustotu $f(x, \theta)$, kde θ je neznámý jednorozměrný parametr, který patří do neprázdného otevřeného intervalu Θ. Potom sdruženou funkci rozdělení výběru při pevných výběrových hodnotách $X_i = x_i, i = 1, 2, \ldots, n$, jako funkci parametru θ, nazveme funkcí věrohodnosti a označíme $L(\mathbf{x}, \theta)$, pro niž platí

$$L(\mathbf{x}, \theta) = L(x_1, x_2, x_3, \ldots, x_n; \theta) = \prod_{i=1}^{n} f(x_i, \theta). \tag{1.3}$$

Základním principem metody maximální věrohodnosti je, že za odhad neznámé hodnoty parametru θ zvolí hodnota $\hat{\theta}$, která při daných realizovaných hodnotách proměnných maximalizuje funkci věrohodnosti. Pokud existuje takový bod $\hat{\theta} \in \Theta$, že pro všechny $\theta \in \Theta$ platí

$$L(\mathbf{x}, \theta) \leq L(\mathbf{x}, \hat{\theta}), \tag{1.4}$$

budeme potom $\hat{\theta}$ nazývat maximálně věrohodným odhadem neznámé hodnoty parametru θ. Pokud má funkce věrohodnosti $L(\mathbf{x}, \theta)$ pro každé $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)'$ derivace podle θ

$$\frac{\partial L(\mathbf{x}, \theta)}{\partial \theta}, \tag{1.5}$$

potom $\hat{\theta}$ musí být řešením rovnice

$$\frac{\partial L(\mathbf{x}, \theta)}{\partial \theta} = 0. \tag{1.6}$$

V literatuře se zpravidla namísto věrohodnostní funkce $L(\mathbf{x}, \theta)$ maximalizuje její přirozený logaritmus. Ten je pak nazýván logaritmickou funkcí věrohodnosti. Věrohodnostní rovnice má tedy tvar

$$\frac{\partial \ln L(\mathbf{x}, \theta)}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial \ln f(x_i, \theta)}{\partial \theta} = 0. \tag{1.7}$$

Souvislost mezi postačující statistikou parametru θ a $\hat{\theta}$ je taková, že pokud je odhad maximální věrohodnosti jediný, potom je to funkce minimální postačující statistiky. Pokud je odhadů maximální věrohodnosti více, můžeme vybrat odhad maximální věrohodnosti jako funkci minimální postačující statistiky.

Ve statistice je test poměru věrohodnosti používán ke srovnání shody dvou modelů, kdy jeden je vložený mezi ostatní. Často je tento přístup využíván, když testujeme, zda je předpoklad pro model platný tzn. když předpokládáme, že dva a více parametrů jsou podobné [9, str. 94-96].

1.3.4 Normální (Gaussovo) rozložení

Gaussovo, neboli normální rozložení, je jedním z nejužívanějších rozložení spojitých náhodných veličin používaných v pravděpodobnostních úlohách. Označuje se $N(\mu, \sigma^2)$. V tomto modelu je předpokládána nulová střední hodnota a časově proměnný rozptyl koeficientů, což prakticky znamená, že se ke konstantní střední hodnotě $\mu \in (-\infty, \infty)$ přičítá velké množství nezávislých náhodných veličin kolísajících nepatrně kolem nuly. Vzniká tak variabilita, která je popsána směrodatnou odchylkou $\sigma \in (0, \infty)$. Náhodná veličina $\mathbb{X}$ má normální rozložení, jestliže má funkci hustoty

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1.8}$$

a distribuční funkci

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt. \tag{1.9}$$

Pro náhodnou veličinu $\mathbb{X}$ s normálním rozložením dále platí, že

$$E(\mathbb{X}) = \mu, \tag{1.10}$$

$$D(\mathbb{X}) = \sigma^2. \tag{1.11}$$

Grafem funkce $f(x)$ je tzv. Gaussova křivka, která je symetrická kolem bodu $x = \mu$ a hodnota tohoto maxima je $\frac{1}{\sigma\sqrt{2\pi}}$. Konstanta μ je střední hodnota a udává polohu maxima, a tedy i celé křivky. Inflexní body jsou $x = \mu - \sigma$ a $x = \mu + \sigma$ [1, str. 44, 48-49][9, str. 62-63].

Z rovnic 1.10, 1.11 a 1.1 usuzujeme, že se jedná o nevychýlené odhady.

Pokud jsou všechna konečněrozměrná rozdělení procesu $\{X_t; t \in T\}$ normální, nazýváme jej Gaussovským náhodným procesem, tzn. že libovolná n-tice hodnot $(X_1, X_2, X_3, \ldots, X_n)$ je náhodným vektorem s Gaussovským náhodným rozdělením.

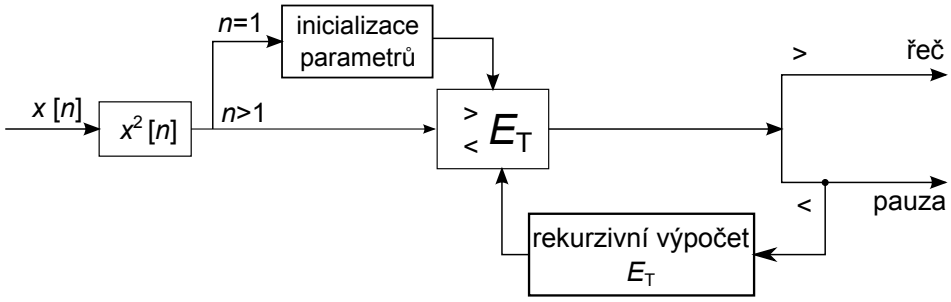
1.4 Metody detekce řečové aktivity

1.4.1 Energetický detektor

Energetický detektor řečové aktivity je založený na sledování energie E_x v krátkých časových úsecích vstupního signálu $x[n]$ a to jak v doméně časové, tak frekvenční. Výpočet energie signálu pro jeden časový úsek je uveden v rovnici 1.12

$$E_x = \sum_{n=m\cdot N}^{m\cdot N+N-1} x^2[n], \tag{1.12}$$

kde m je pořadí aktuálně zpracovaného segmentu a N je délka segmentu [18]. Uvedený vzorec 1.12 pro výpočet energie signálu v daném segmentu je aplikovanou



Obr. 1.2: Schéma energetického detektoru řečové aktivity.

formou obecného vzorce

$$E = \frac{1}{2\pi} \int_{-\pi}^{\pi} S(\mathrm{e}^{\mathrm{j}\theta}) d\theta, \tag{1.13}$$

kde $S(\mathrm{e}^{\mathrm{j}\theta})$ je energetická spektrální hustota vstupního signálu a θ je normalizovaný kmitočet v radiánech [16].

Pro odhad rozhodovacího prahu je nejprve pro vypočtenou hodnotu energie použito vyhlazení E_T. Rozumíme tím adaptivní algoritmus, který v segmentech, kde není přítomná hlasová aktivita, počítá vyhlazenou hodnotu energie signálu podle rovnice 1.14

$$E_T[m] = \mathrm{p} E_T[m-1] + (1-\mathrm{p}) E_x[m], \tag{1.14}$$

kde p je konstanta odpovídající zvolené velikosti okna, přičemž $\mathrm{p} = 1/\mathrm{N}$. Takto vypočtená vyhlazená energie signálu se použije pro výpočet prahové hodnoty

$$E_t = E_T + \mathrm{k} \tag{1.15}$$

Konstanta k je subjektivně zvolená konstanta, jejíž hodnota je určena na základě experimentálních pokusů. Nyní jsou k dispozici již všechny prostředky pro rozhodnutí, zda se jedná a řečově aktivní segment. Pokud je hodnota energie aktuálně vypočteného segmentu větší než práh, který jsme získali z rovnice 1.15, jedná se o řečově aktivní segment, tzn. $E > E_t$. V opačném případě se jedná o segment hlasově neaktivní, když $E < E_t$ [16].

1.4.2 Statistické metody detekce

V minulosti byl pro odhad spektrální amplitudy řečového signálu zaveden algoritmus spektrálního odečítání, kde byl odhad krátkodobé spektrální amplitudy počítán jak druhá mocnina odhadu maximální věrohodnosti rozptylu každé spektrální komponenty. Další používaný přístup byl založen na Wienerově filtraci, kde byl odhad krátkodobé spektrální amplitudy získán jako modul optimální minimální střední

kvadratické chyby každé spektrální komponenty signálu. Ani jeden z těchto přístupů však nesplňuje požadavky pro optimální odhad spektrální amplitudy v úvahu s předpokládaným statistickým modelem.

Druhý uvedený přístup byl následně použit jako základ k nové metodě, která předpokládá znalost a priori koeficientů Fourierovy transformace. Protože je neznáme, musíme je odhadnout na základě převzatého statistického modelu. Řečový signál není ani stacionární ani ergodický proces. Můžeme taktéž předpokládat stejné vlastnosti hluku na pozadí, nemusí tomu však vždy být tak. Ztrácíme tím možnost analýzy signálu z dlouhodobého hlediska a musíme brát v úvahu krátké úseky, u nichž se dá alespoň částečná stacionarita předpokládat.

Pro modelování statisticky nezávislých koeficientů byl proto zvolen Gaussovský model, ve kterém je odhad parametrů založen na rozhodnutím řízeném přístupu [15]. Na těchto detektorech byla prokázána větší přesnost detekce než na detektorech založených na konvenčních metodách i přesto, že konfigurace detektoru je závislá pouze na několika málo parametrech. Tyto algoritmy operují hlavně v doméně diskrétní Fourierovy transformace [3].

Statistické metody předpokládají, že dočasné výchylky statistických vlastností hluku na pozadí jsou mnohem menší než u řeči, což umožňuje odhadovat časově proměnné statistiky hluku na pozadí navzdory nahodilému výskytu řečového signálu. V minulosti bylo provedeno spousta pokusů měření distribuce pravděpodobnosti řečového signálu, ale i přesto se zdá, že ideální model je nedosažitelný [3][14].

Rozhodnutí na základě odhadu maximální věrohodnosti

Rozhodnutí o tom, zda je zkoumaný segment signálu pouze šum nebo šum s řečovým signálem může být provedeno na základě statistického rozhodnutí, kdy je měřen rozdíl mezi hlukem na pozadí a pozorovaným signálem a rozhodnutím na základě stanoveného prahu, který je často definován empiricky. Jednou z možností, jak statistické pravidlo rozhodnutí stanovit je zobecněný test poměru věrohodnosti a to za předpokladu, že statistiky hluku na pozadí jsou dány a priori odhadem těchto statistických vlastností.

Bude použit statistický model ve kterém řeč i hluk na pozadí jsou Gaussovské náhodné procesy, které jsou navzájem nezávislé. Potom koeficienty diskrétní Fourierovy transformace každého procesu jsou asymptoticky nezávislé Gaussovské náhodné proměnné [14].

Odhad maximální věrohodnosti je odvozen pro rozptyl k-té spektrální komponenty signálu n-tého analyzovaného rámce o délce L. Odhad je založen na L pozorování $Y_k(n) = \{Y_k(n), Y_k(n-1), \ldots, Y_k(n-L+1)\}$, u kterých předpokládáme,

že jsou statisticky nezávislé. Toto tvrzení platí pro nulový přesah rámců, který je použit kvůli jednoduchosti modelování a popisu.

Je nastavena hypotéza, že zpracovávaný rámec obsahuje buď signál, kde je pouze hluk na pozadí

$$H_1 : x[n] = r[n], \tag{1.16}$$

a nebo obsahuje signál řečový s hlukem na pozadí

$$H_1 : x[n] = s[n] + r[n]. \tag{1.17}$$

Pro použité signály $x[n]$, $s[n]$ a $r[n]$ vypočteme jejich diskrétní Fourierovy obrazy $\mathbf{X}$, $\mathbf{S}$, $\mathbf{R}$. Potom hypotézy nastavené pro obrazy signálů ve spektrální doméně jsou

$$H_0 : \quad \mathbf{X} = \quad \mathbf{R} \tag{1.18}$$

$$H_1 : \quad \mathbf{X} = \quad \mathbf{S} + \mathbf{R}. \tag{1.19}$$

Na základě Gaussova statistického modelu a předpokladu asymptotické nezávislosti spektrálních komponent jako Gaussovských náhodných proměnných můžeme funkci hustoty pravděpodobnosti pro jednotlivé hypotézy vyjádřit pomocí rovnic [14]

$$p(\mathbf{X}|H_0) = \prod_{k=0}^{L-1} \frac{1}{\pi\lambda_R(k)} \exp\left(-\frac{|X_k|^2}{\lambda_R(k)}\right) \tag{1.20}$$

$$p(\mathbf{X}|H_1) = \prod_{k=0}^{L-1} \frac{1}{\pi[\lambda_S(k) + \lambda_R(k)]} \exp\left(-\frac{|X_k|^2}{\lambda_S(k) + \lambda_R(k)}\right). \tag{1.21}$$

Jako $\lambda_S(k)$ a $\lambda_R(k)$ jsou označeny rozptyly k-té spektrální komponenty S_k resp. R_k. Tyto rozptyly jsou pomalu se měnící parametry, jejichž minimálně se měnící průběh lze považovat za konstantní v rozsahu L pozorování [3].

Pro rozhodnutí založené na statistice, zda se jedná o signál kde je pouze hluk na pozadí nebo i řeč, použijeme test poměru věrohodnosti. Pro k-té frekvenční pásmo jej vypočítáme pomocí výrazu [5]

$$\Lambda_k = \frac{p(X_k|H_1)}{p(X_k|H_0)}, \tag{1.22}$$

kde Λ_k vyjadřuje věrohodnostní poměr k-tého frekvenčního pásma. Rozhodovací pravidlo pro detektor řečové aktivity je zavedeno jako geometrický průměr poměrů věrohodností počítaných pro jednotlivé frekvenční pásma:

$$\ln \Lambda = \frac{1}{L} \sum_{k=0}^{L-1} \ln \Lambda_k \gtrless_{H_0}^{H_1} \eta. \tag{1.23}$$

η v této rovnici vyjadřuje práh pro rozhodování.

Nyní je třeba definovat výraz pro a priori SNR, které se v každém analyzovaném rámci odhaduje znovu z důvodu nestacionarity řečového signálu. Pro tento účel je využit odhad maximální věrohodnosti rozptylu spektrálního obrazu řeči. Tento přístup předpokládá znalost rozptylu spektrálního obrazu hluku na pozadí. V praxi se pro odhad rozptylu hluku používají rámce, ve kterých není detekována řeč a jsou v časové ose nejblíže analyzovanému oknu. Pokud je hluk na pozadí stacionární, stačí jeho vlastnosti odhadnout pouze jednou, většinou z prvního rámce signálu bez řeči. Odhad a priori SNR je podle [3] roven

$$\xi_k = \frac{\lambda_S(k)}{\lambda_R(k)}. \tag{1.24}$$

Vztah pro výpočet a posteriori SNR odvozen v [3] je

$$\gamma_k = \frac{|X_k|^2}{\lambda_R(k)}. \tag{1.25}$$

Dosazením rovnice 1.20 a 1.21 do rovnice 1.23 a využitím vztahů 1.24 a 1.25 je možno poměr věrohodnosti vyjádřit výrazem

$$\Lambda_k = \frac{1}{1+\xi_k} \exp\left(\frac{\gamma_k \xi_k}{1+\xi_k}\right). \tag{1.26}$$

Protože rozptyl hluku $\lambda_R(k)$ už z předchozích pozorování je znám, dostaneme tak odhad maximální věrohodnosti

$$\hat{\xi_k} = \gamma_k - 1. \tag{1.27}$$

Rovnice 1.27 je poté dosazena do vztahu 1.26 [15]

$$\ln \hat{\Lambda} = \frac{1}{L} \sum_{k=0}^{L-1} (\gamma_k - \ln \gamma_k - 1). \tag{1.28}$$

Konečná podoba výpočtu poměru věrohodnosti po dosazení vztahu 1.25 do rovnice 1.28 má tvar [15][17]

$$\hat{\Lambda} = \frac{1}{L} \sum_{k=0}^{L-1} \left(\frac{|X_k|^2}{\lambda_R(k)} - \ln \frac{|X_k|^2}{\lambda_R(k)} - 1\right). \tag{1.29}$$

Adaptivní algoritmus pro výpočet rozptylu spektra hluku na pozadí

Optimální odhad rozptylu spektrálních koeficientů hluku na pozadí $\lambda_R(k)$ v podmínkách minimální střední kvadratické chyby je definován

$$\hat{\lambda}_R(k) = E(\lambda_R(k)|X_k) = E(\lambda_R(k)|H_0)P(H_0|X_k) + E(\lambda_R(k)|H_1)P(H_1|X_k). \tag{1.30}$$

Za použití Bayesova pravidla, definovaného např. v [1, str. 17], můžeme odvodit

$$P(H_0|X_k) = \frac{p(X_k|H_0)P(H_0)}{p(X_k|H_0)P(H_0) + p(X_k|H_1)P(H_1)} = \frac{1}{1+\epsilon\Lambda(k)}, \tag{1.31}$$

kde $\Lambda(k)$ dosadíme z rovnice 1.22 a ϵ spočítáme jako

$$\epsilon = \frac{P(H_1)}{P(H_0)}. \tag{1.32}$$

Podobně je odvozen vztah

$$P(H_1|X_k) = \frac{\epsilon\Lambda(k)}{1+\epsilon\Lambda(k)}. \tag{1.33}$$

Po dosazení rovnic 1.33 a 1.32 do vztahu 1.31 dostaneme

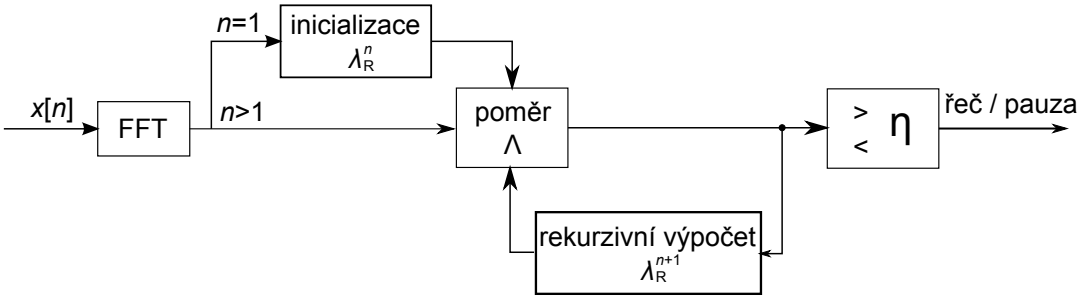
$$E(\lambda_R(k)|X_k) = \frac{1}{1+\epsilon\Lambda(k)}E(\lambda_R(k)|H_0) + \frac{\epsilon\Lambda(k)}{1+\epsilon\Lambda(k)}E(\lambda_R(k)|H_1). \tag{1.34}$$

Protože je odhad rozptylu spektrálních koeficientů šumu na pozadí prováděn pro každý rámec, bude nutné zavést proměnnou n, která určuje kolikátý rámec je právě zpracováván. Pro snadnější dosažení odhadu $\lambda_R(k)$ je nahrazen v rovnici 1.34 výraz $E(\lambda_R^n(k)|H_0)$ absolutní hodnotou spektrálních složek z aktuálního měření $|X_k^n|$ pokud se jedná o rámec bez řeči. V případě, že je přítomna řeč nahradíme výraz $E(\lambda_R^n(k)|H_1)$ odhadem z předchozího rámce $\hat{\lambda}_R^n(k)$.

Protože již není výpočet prováděn pro všechny spektrální koeficienty, nahradíme výpočet $\Lambda(k)$ geometrickým průměrem všech složek Λ^n. Výsledný vztah pro výpočet odhadu aktuálního rozptylu spektrálních koeficientů hluku na pozadí má pak tvar

$$\hat{\lambda}_R^n(k) = \frac{1}{1+\epsilon\Lambda^n}|X_k^n|^2 + \frac{\epsilon\Lambda^n}{1+\epsilon\Lambda^n}\hat{\lambda}_R^{n-1}. \tag{1.35}$$

Uvedený vztah popisuje odhad časově variantního spektra s exponenciálním průměrováním. Rovnice 1.35 naznačuje, že čím blíže je pozorované spektrum signálu k aktuálnímu odhadu, tím rychlejší je rychlost konvergence adaptačního systému. Tuto rychlost konvergence je možno nastavit parametrem ϵ. Díky použití rozhodovací informace Λ je možné spektrum hluku nastavovat pokud se vyskytuje v signálu řeč nebo ne. Pro přesnější odhad parametrů šumu počítáme odhad jeho rozptylu pro každý kmitočet [14][17].



Obr. 1.3: Schéma statistického detektoru řečové aktivity.

1.4.3 Výpočet odstupu signálu od šumu

Detektor řečové aktivity potřebuje neustálou analýzu hluku na pozadí, jehož úroveň se mění a s tím související odstup od užitečného signálu řeči. Z toho důvodu je zaveden následující postup pro výpočet SNR. Protože u testovacích nahrávek známe úroveň signálu řečového i signálu hluku na pozadí zvlášť, můžeme díky tomu při následném součtu signálů pracovat s předem známou informací o jejich úrovni a požadované SNR tak dopředu vypočítat vztahem

$$\mathrm{SNR} = 10 \log \frac{\sum_{n=0}^{L-1} s^2[n] \cdot vad[n]}{\sum_{n=0}^{L-1} r^2[n] \cdot vad[n]}, \tag{1.36}$$

kde $s[n]$ a $r[n]$ jsou signály řeči, resp. hluku na pozadí, L je délka zpracovávaného okna a $vad[n]$ je informace, zda je zpracovávaný vzorek signálu n součástí segmentu řeči nebo bez řeči. Z rovnice je zřejmé, že úroveň SNR počítáme výhradně ze segmentů, kde je přítomná řeč [20].

1.4.4 Ohodnocení detektoru řečové aktivity

Při reálném provozu detektoru řečové aktivity můžou nastat čtyři situace jako výsledek jeho aktivity. Správným rozhodnutím detektoru může být úspěšná detekce řeči (angl. *true positive*) nebo úspěšná detekce pauzy (*true negative*). Dále můžou nastat dvě situace chybných rozhodnutí detektoru a to buď chybná detekce řeči (*false positive*), nazývaná taktéž falešný alarm (*false alarm*) nebo může detektor chybně ohodnotit příchozí signál jako pauzu (*false negative*).

Jednotlivé stavy se vyjadřují v poměru k délce řečového nebo neřečového signálu v procentech. Pro úsek, ve kterém je ve skutečném signálu přítomná řeč, může být výsledek VADu ohodnocen pouze jako *true positive* nebo *false negative*. Pokud je signál ve skutečnosti pouze hluk na pozadí, výsledek detekce může být pouze *true negative* nebo *false positive*. Jednotlivé poměry lze vypočítat podle následujících rovnic

$$TP = (N_{TP}/S) \cdot 100\,[\%] \tag{1.37}$$

$$FN = (N_{FN}/S) \cdot 100\,[\%] \tag{1.38}$$

pro signál v rámci oblastí, kde je přítomná řeč i hluk na pozadí nebo

$$TN = (N_{TN}/R) \cdot 100\,[\%] \tag{1.39}$$

$$FP = (N_{FP}/R) \cdot 100\,[\%], \tag{1.40}$$

Procento správně detekované řeči (TP - True Positive)	Procento chybně detekované řeči (TN - True Negative)
Procento chybně detekované pauzy (FN - False Negative)	Procento správně detekované pauzy (FP - False Positive)

Tab. 1.1: Vyhodnocení výsledků detektoru

pro oblasti, kde je detekován pouze hluk. N_{xx} je počet vzorků, ve kterých je signál vyhodnocen jedním ze čtyř možných výsledků a S, R je počet vzorků signálu, ve kterých je přítomná řeč s hlukem, resp. pouze hluk. Výsledky se pro přehlednost sepisují do tabulky jak je uvedeno v tabulce 1.1. Pro kontrolu správnosti výsledků musí být celkový součet v jednotlivých sloupcích roven 100%.

Tyto výsledky mohou být reprezentovány graficky pomocí ROC křivky (Receiver operating characteristics). Na osu x jsou vyneseny jednotlivé procentuální hodnoty chybně detekované řeči a na osu y jsou vyneseny procentuální hodnoty správně detekované řeči. Systém detekce řečové aktivity dosáhne při různých vstupních parametrech rozdílných výsledků detekce. Jejich hodnoty TP a FP jsou následně vyneseny do grafu, čímž vznikne křivka z levého spodního rohu grafu k pravému hornímu rohu.

Čím lepší bude systém v detekci řečové aktivity, tím více se bude tato křivka přibližovat levému hornímu rohu. V bodu křivky, který je levému hornímu rohu grafu nejblíže dosáhne systém detekce řečové aktivity nejlepších výsledků. ROC křivka ideálního detektoru by potom měla průběh stejný jako osa y a v maximu ($y = 100$) by pokračovala jako rovnoběžka s osou x při maximální hodnotě osy y [12].

1.5 Databáze nahrávek pro VAD

Při tvorbě databáze nahrávek, která slouží pro testování detektoru řečové aktivity nemůžeme nikdy pokrýt všechny reálné situace. Proto se musíme omezit na simulaci nejběžnějších situací, ve kterých bude VAD využíván. Rozsáhlou a kvalitní databázi je možné vytvořit pro kvalitní ověření funkčnosti detektoru, ovšem za cenu velké časové náročnosti, nákladné realizace a velkého datového objemu. Z toho důvodu je efektivnější vytvořit menší databázi, která bude zachycovat jak prostředí typické pro využití detektoru, tak fonémy typické pro jazyk, pro který je detektor určen. Zvláštní pozornost by měla být věnována problematickým jevům pro jednotlivé jazyky a zejména otestování slabých míst detektoru řečové aktivity [6].

Nahrávky řeči pro naši databázi byly vytvořeny v bezodrazové komoře v laboratoři PA-327 na Ústavu telekomunikací. Testovací nahrávky slouží pouze pro ověření funkčnosti detektoru řečové aktivity, proto byly zvoleny čtyři věty, které byly nahrány:

- „Cíl - horizont 200 metrů vpravo“,
- „Tank na třech hodinách, vzdálenost 500 metrů“,
- „Řidič vpřed“,
- „Zde Alfa 1, Alfa 1, Alfa 1, volá Deltu 2“.

Věty byly zvoleny z vojenského prostředí, pro které je VAD navrhován.

K nahrávání byl použit nahrávací přístroj Tascam HD-P2, zesilovač mikrofonního signálu NEXON (inv.č. 314762) a mikrofon Brüel & Kjær 4189 (inv.č. 1000153771).

Pro tyto nahrávky je důležité označení počátku a konce řečové aktivity pro účely testování. Pro rozsáhlou databázi nahrávek je nejvýhodnější použít označení pomocí jednoduchého energetického detektoru. Protože jsou nahrávky pořízeny v bezodrazové komoře s prakticky nulovým okolním hlukem a nulovými odrazy, je tato metoda efektivní a spolehlivá. Jelikož obsahuje naše databáze pouze malé množství vzorků, bylo toto značení nahrávek provedeno ručně. Ruční značení je nejspolehlivější metodou, při které by neměly vzniknout neočekávané chyby.

Jako hluk na pozadí byly dodány nahrávky hluku zevnitř tří motorových pásových a kolových transportérů.

1.6 Segmentace nahrávek

V rámci této diplomové práce jsou použity pouze algoritmy, které zpracovávají vstupní signál po segmentech. Délka segmentů se pohybuje od 10ms do 30ms, čemuž při vzorkovací frekvenci $f_{vz} = 8000\,\mathrm{Hz}$ odpovídají délky segmentů od 80 vzorků do 240 vzorků. V tak dlouhých úsecích lze řeč považovat za kvazistacionární (částečně stacionární), délka segmentů odpovídá maximálním fyziologickým možnostem hlasového traktu pro změnu hlásky a tudíž i pro změnu vlastností řeči.

Počet segmentů je možné vypočítat podle vzorce

$$K_{\mathrm{L}} = \left\lceil \frac{N - P}{L - P} \right\rceil, \tag{1.41}$$

kde N značí celkovou délku vstupního signálu ve vzorcích, L délku segmentu ve vzorcích a P určuje velikost přesahu ve vzorcích. Závorky $\lceil\rceil$ způsobují zaokrouhlení výsledku směrem nahoru.

Při volbě přesahu P musí být brán zřetel na délku segmentu L. Délka přesahu musí být zvolena v intervalu $< 1; L - 1 >$, aby nedošlo k chybě výpočtu. Prakticky jsou však používány hodnoty s větším odstupem od okrajů intervalu [13, str. 11-20].

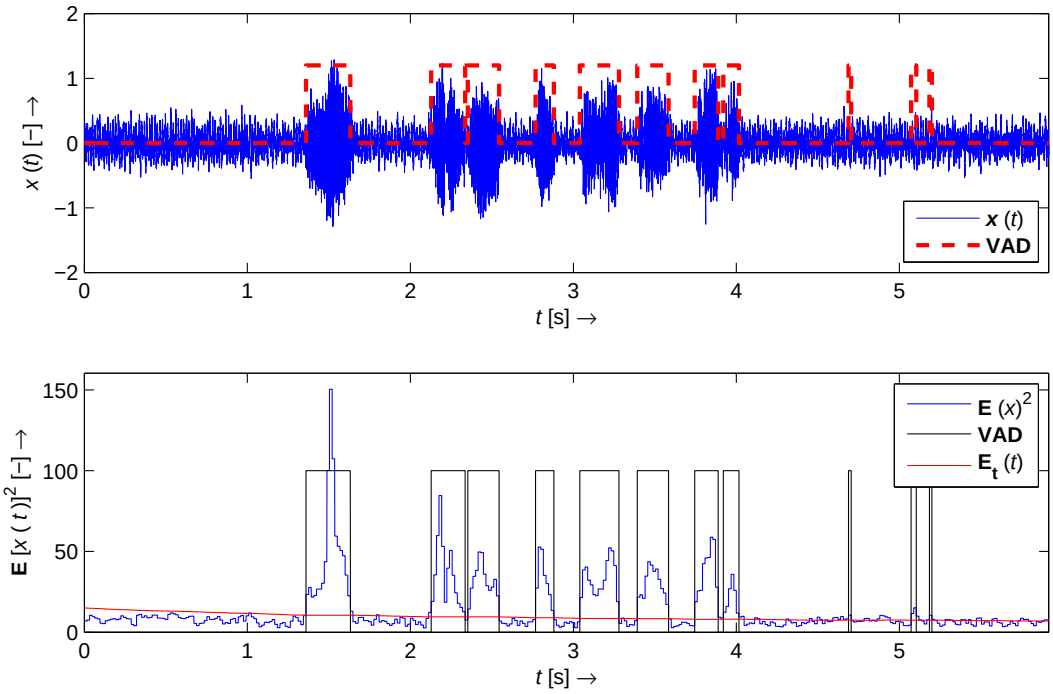
2 REALIZACE VAD V PROSTŘEDÍ MATLAB

Funkce energetického i statistického detektoru řečové aktivity byla ověřena na základě teoretického rozboru v první části textu. Realizace byla provedena v prostředí Matlab. Popis realizace je uveden v následujících kapitolách i s názornými ukázkami funkčnosti jednotlivých metod.

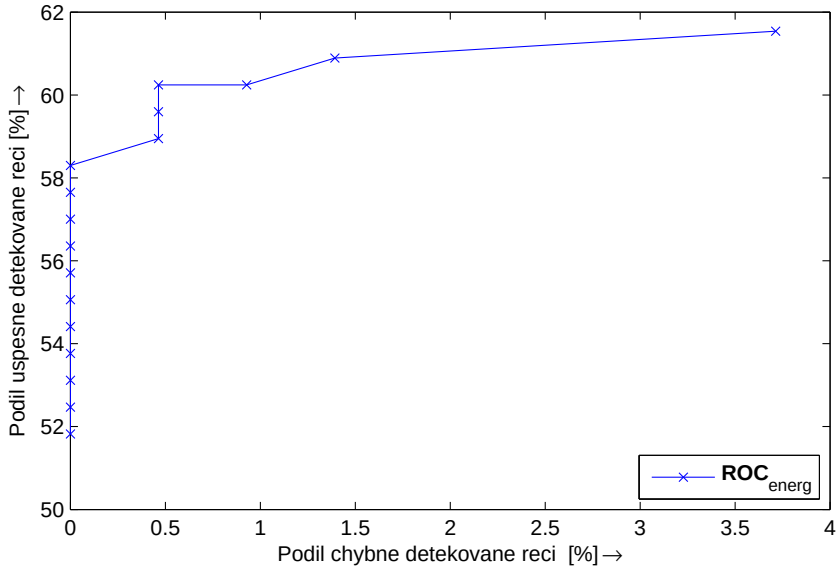
2.1 Ověření funkce energetického detektoru

Pro realizaci energetického detektoru řečové aktivity byl zvolen systém teoreticky popsaný v kapitole 1.4.1. Výpočet energie v čase byl proveden podle vzorce 1.12. Zvlášť důležitým parametrem pro energetický VAD je volba délky segmentu, který je zpracováván. Po několika experimentech bylo zjištěno, že tento typ detektoru je velmi citlivý na velikost okna, ze kterého se energie signálu počítá. Změna výsledků je při změně délky okna znatelná a pro opětovné dosáhnutí funkčnosti je nutný zásah do parametrů detektoru.

Pro energetický detektor se jako nejvhodnější osvědčily délky okna 128 a 256



Obr. 2.1: Detekce řeči pomocí energetického detektoru při $N = 256$ s přesahem $0,5$; SNR $= 6\,\mathrm{dB}$, $E_T = 15$, k $= 8$ a p $= 0,989$.



Obr. 2.2: ROC křivka pro různou konstantu $k =< 0; 10 >$, SNR = 6 dB, $E_T = 15$, $p = 0,989$.

s přesahem 50%. Na obrázku 2.1 je v horním grafu vidět oblasti, které detekoval energetický VAD. Délka okna byla zvolena $N = 256$ s přesahem $0,5$. Signál, na který byl detektor testován, je složen z hluku pásového transportéru a věty „Cíl - horizont 200 metrů vpravo“ s tím, že řeč má od hluku automobilu odstup 6 dB. Výchozí hodnota pro práh byla zvolena $E_T = 15$, hodnota konstanty k ze vzorce 1.15 byla nastavena na hodnotu 8 a hodnota konstanty pro vyhlazování prahu z rovnice 1.14 byla zvolena $p = 0,989$. Ve spodním grafu je vidět, že celková energie vypočtená pro jednotlivé segmenty (vyznačena modře) kopíruje ve svých špičkách detekovanou řeč zejména v oblasti, kde se vyskytují energeticky bohaté samohlásky. Naopak v místech, kde se nachází souhlásky s nízkou energií zaniká energie signálu v hluku na pozadí a řečová aktivita tak není detekována, což způsobuje chybu *false negative*.

Ke konci testovacího signálu, kdy se již nevyskytuje řečová aktivita, klesá hodnota prahu vlivem adaptace a pohybuje se na úrovni energie signálu. Proto byla zvolena konstanta k = 8 aby bylo zamezeno chybným detekcím řeči v těchto oblastech. Parametry detektoru byly určeny na základě vykreslení ROC křivek pro proměnlivé hodnoty jednotlivých parametrů. Na obr. 2.2 je vidět křivka pro různé hodnoty konstanty k pro SNR = 6 dB, základní $E_T = 15$ a $p = 0,989$. Koeficient k byl postupně volen v intervalu $< 0; 10 >$ z něhož bylo generováno 50 rovnoměrně rozložených hodnot. Z grafu 2.2 byla odečtena ideální hodnota pro konstantu $k = 1,02$ v bodě $[0,4642; 60,2429]$.

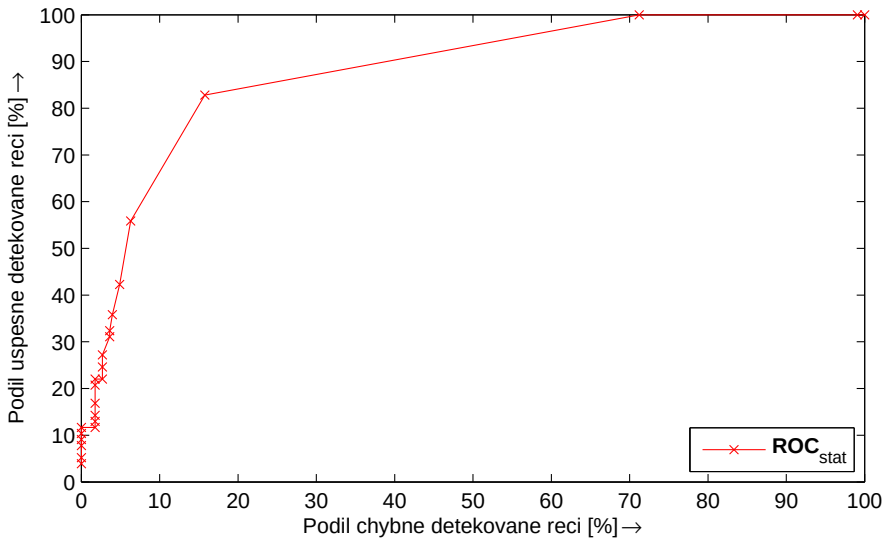
2.2 Ověření funkce statistického detektoru

Statistický VAD byl zrealizován podle teoretického rozboru v kap. 1.4.2. Základní důležitou volbou je opět délka zpracovávaného okna. Několika pokusy bylo zjištěno, že s klesající délkou okna se zmenšuje podíl chybně detekované řečové aktivity (až o 20 %), ale klesá též podíl úspěšně detekované řeči a to o přibližně 15 %. Uvedené hodnoty poklesu platí pro změnu velikosti okna z 512 vzorků na 128 vzorků. Z toho důvodu byla pro ověření funkčnosti zvolena délka okna 512 vzorků s přesahem $0,5$.

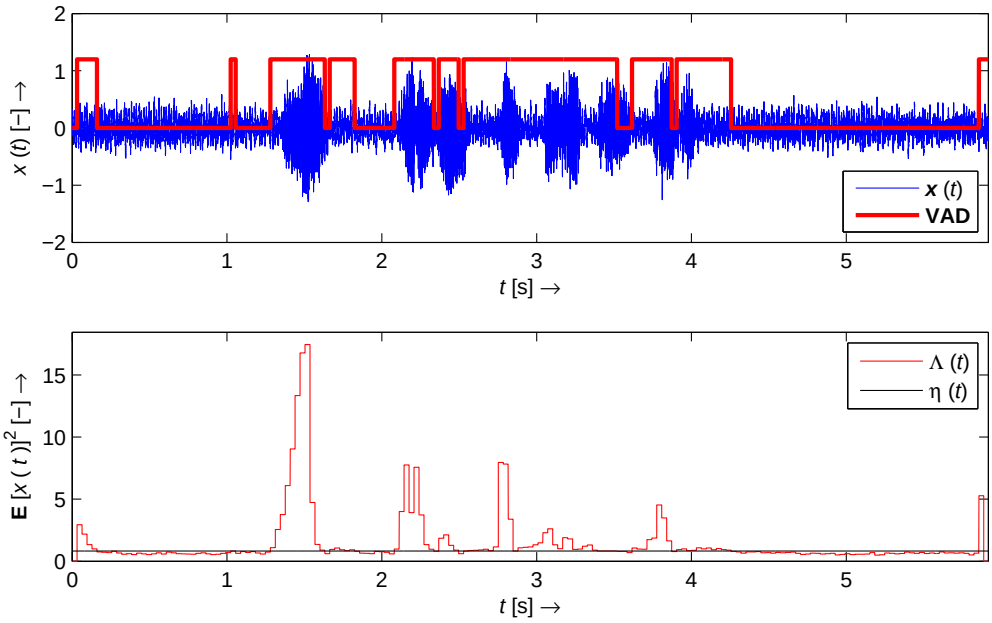
Statistický detektor předpokládá, že první testovaný segment signálu je pouze hluk bez řeči, proto v něm dochází k výpočtu statistických vlastností tohoto hluku na pozadí a dál je při výpočtu pouze korigována tato hodnota. Detektor předpokládá rozdělení statistických vlastností podle normálního rozložení a z něj byl odvozen vztah pro vyhodnocení segmentu jako řeči nebo pauzy na základě metody maximální věrohodnosti podle rovnice 1.29.

Pro rozhodnutí, zda je signál pauza nebo řeč je definována konstanta η. Na základě pokusu, kdy při odstupu signálu řeči od šumu 6 dB byla testována úspěšnost detekce pro 50 hodnot konstanty $\eta =< 0; 10 >$, byla jako ideální odečtena z grafu v bodě se souřadnicemi $[15,76; 82,82]$ hodnota $\eta = 0,8163$. Průběh ROC křivky je možno vidět na obr. 2.3.

Další konstantou, kterou je třeba určit, je poměr pravděpodobností výskytu řeči ku pauze ϵ. Její hodnota byla odvozena stejným způsobem jako konstanta η a nejlepší hodnoty dosáhla v bodě $[14,8286; 89,3016]$ $\epsilon = 6,7368$. Na obr. 2.4 je možno vidět,



Obr. 2.3: ROC křivka pro různou konstantu $\eta =< 0; 10 >$, SNR = 6 dB.



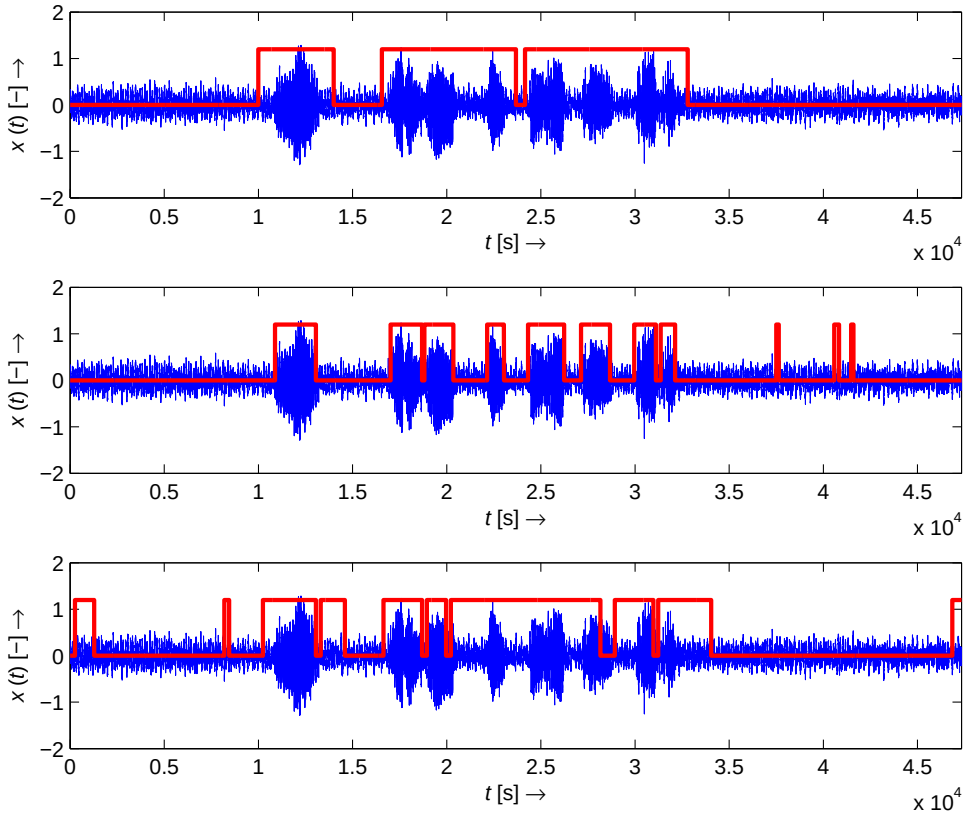
Obr. 2.4: Výsledky detekce statistického detektoru pro $\eta = 0,8163$, SNR $= 6\,\mathrm{dB}$, $\epsilon = 6,7368$.

jak detekoval implementovaný statistický detektor řečové aktivity průběh vstupního signálu. O poznání lépe detekoval oblasti, kde jsou souhlásky se slabým energetickým spektrem, které se těžko detekuje na energetickém detektoru. Ovšem statistické změny spektra signálu už jsou výraznější, což ukazuje spodní graf. Na signálu se objevilo několik falešně detekovaných segmentů. Tato chyba je způsobena nutností adaptace algoritmu na počáteční podmínky v úvodu signálu.

2.3 Porovnání výsledků obou detektorů

Pro porovnání úspěšnosti detekce jsou na obr. 2.5 zobrazeny v prvním grafu ručně označené segmenty řeči, na základě kterých se počítají poměry pro ROC. V dalších dvou grafech jsou zobrazeny průběhy detekce řeči u energetického a statistického detektoru. Jejich počáteční parametry byly nastaveny z hodnot, pro něž dosahovaly hodnoty v předchozích kapitolách nejlepší výsledky. Jak je vidět, energetický detektor není schopen detekovat části slov (především souhlásky), které mají malou energetickou vydatnost. Taktéž je citlivý na náhlé výkyvy v hluku vozidla na pozadí ke konci signálu, kde chybně označil několik segmentů jako řeč.

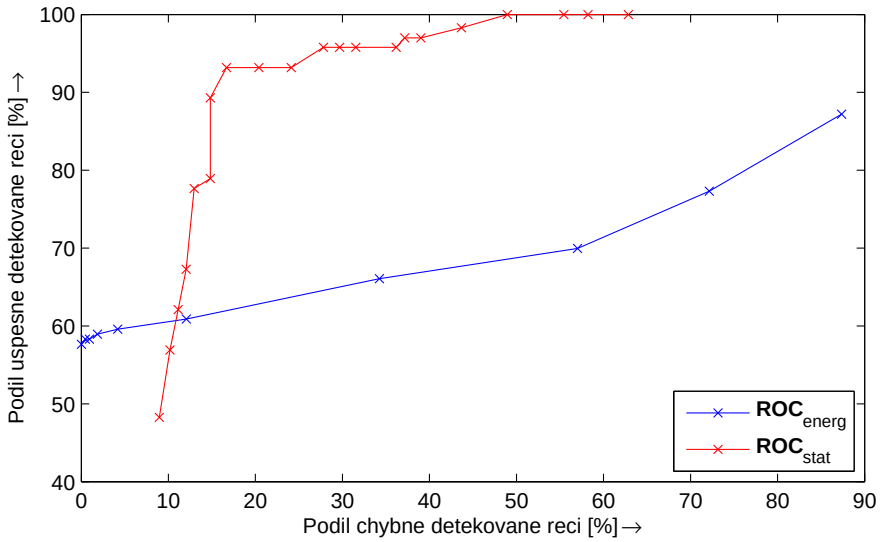
Naproti tomu má statistický detektor několik chybně detekovaných segmentů v úvodu signálu při adaptaci algoritmu, ovšem následně vykazuje lepší úspěšnost



Obr. 2.5: Výsledky detekce ideálního (nahoře), energetického (uprostřed) a statistického detektoru.

při detekci souhlásek. Celkově je oblast detekované řeči větší než u energetického detektoru, což výrazně přispívá ke srozumitelnosti řeči.

Následující graf na obr. 2.6 ukazuje výkonnost obou typů detektorů pro různé hodnoty odstupu signálu řeči od hluku vozidla na pozadí. Odstup byl zvolen v rozmezí od 0 dB do 20 dB se skokem po 1 dB. Z grafu je jasně vidět, že statistický detektor prokazuje lepší podíl správně detekovaných úseků řeči, zatímco detekuje minimální množství chybně detekované řeči. Maxima výkonnosti dosahuje v bodě $[16,69; 93,19]$, což odpovídá hodnotě 7 dB. Energetický detektor prokazuje lepší výsledky pouze do hodnoty, kde se obě křivky protínají, což odpovídá přibližně odstupu 2 dB a většímu. Zde je možno pozorovat slabinu statistického detektoru. I když by měl řeč s vyšším SNR detekovat lépe, díky nastaveným parametrům, které jsou optimalizovány pro odstup 6 dB se zvyšuje s rostoucím SNR počet chybně detekovaných úseků řeči.



Obr. 2.6: ROC křivky pro různé hodnoty odstupu řeči od hluku vozidla na pozadí $L =< 0; 20 >$ dB.

3 IMPLEMENTACE VAD V JAZYCE C

Oba algoritmy pro detekci hlasové aktivity byly po ověření v prostředí Matlab implementovány v jazyce C. Ten byl zvolen, protože se jedná o nízkoúrovňový univerzální jazyk a zdrojový kód v jazyce C je možné použít i pro signálové procesory. Implementace detektoru hlasové aktivity na signálovém procesoru může být jednou z možných aplikací v praxi.

Systém byl kompilován pomocí kompilátoru gcc verze 4.5.1, k vývoji aplikace bylo používáno vývojové prostředí Eclipse verze 3.6.1 v operačním systému Linux Fedora 14 ve 64-bitové verzi s jádrem Linux kernel verze 2.6.35.13. Všechny verze jsou aktuální a platné k datu 18. 5. 2011. Pro implementaci byly použity dle zadání diplomové práce open-source knihovny a programy [8].

3.1 Deklarované datové struktury pro detektor

Struktury v jazyce C slouží pro seskupení jedné nebo několika proměnných různých typů. Díky nim je možná daleko snažší manipulace se skupinou příbuzných proměnných [7, str. 145]. Pro uložení důležitých hodnot o vstupním zvukovém souboru byla deklarována struktura `soubor` spolu s jejími složkami

```
struct soubor {
   SF_INFO sndInfo; //informace o vstupnim souboru
   double *buffer;  //ukazatel na data vstupniho souboru
   int success;     //informace o vysledku nacteni souboru
};
```

Složka `SF_INFO sndInfo` obsahuje informace o vstupním audio souboru. Její struktura je definována knihovnou *libsndfile* a bude popsána v další části textu v kap. 3.2.1. Ukazatel na pole `buffer` typu `double` obsahuje adresu počáteční buňky paměti pole, ve kterém jsou uloženy jednotlivé vzorky vstupního audio souboru. Této složky je s výhodou využíváno při odkazu na pole hodnot jednotlivých vzorků signálu v různých částech zdrojového kódu. Při volání funkce tak nemusí být použito celé pole ale pouze ukazatel na něj.

Poslední složkou ve struktuře je proměnná `success` typu `int`, kde se ukládá výsledek o načítání vstupního zvukového souboru. Pokud je audio soubor načten úspěšně, je hodnota proměnné nastavena na 1, pokud ne, má hodnotu 0.

Další strukturou je `vad_info`, jejíž složky obsahují informace o segmentaci vstupního souboru, výsledky detekce a procesorový čas nutný k výpočtu detekce. Struktura je deklarována

```
struct vad_info {
   int K;     //pocet segmentu
   int L;     //delka segmentu
   int P;     //presah segmentu
   int *vad;  //ukazatel na pole s vysledky detekce
   double cas_timespec;  //delka vypoctu
   double cas_timeval;   //delka vypoctu
};
```

kde `K` značí počet segmentů v signálu, `L` obsahuje délku jednoho segmentu a `P` určuje přesah jednotlivých segmentů. Všechny tyto proměnné jsou typu `int` a jejich jednotkou jsou vzorky signálu.

Ukazatel na pole celočíselné proměnné `vad` ukazuje na první buňku paměti, ve které se nachází informace o tom, zda byla detekce pro jednotlivé segmenty úspěšná nebo ne. Pole má tedy délku K a nabývá hodnot 0 nebo 1. Pole je alokováno v počátku všech funkcí jednotlivých detektorů hlasové aktivity. Ještě předtím jsou naplněny hodnoty délky segmentu `L`, velikosti přesahu `P` a na základě informací ze struktury `soubor` je vypočítán počet segmentů `K` podle vzorce 1.41. Kompletní naplnění hodnot je povedeno pomocí funkce

```
struct vad_info pocet_seg (struct soubor *my_soubor, int L, int P);
```

Poslední složkou struktury `vad_info` jsou `cas_timespec` a `cas_timeval` typu `double`, do kterých jsou po výpočtu detekce hlasové aktivity uloženy hodnoty délky výpočtu v milisekundách.

3.2 Čtení externích zvukových souborů

Implementovaný detektor hlasové aktivity pracuje s externími zvukovými soubory. Jejich načítání do aplikace je prováděno za pomoci knihovny *libsndfile*. Jedná se o knihovnu s otevřeným zdrojovým kódem šířenou pod licencí Gnu Lesser General Public Licence.

Knihovna je primárně napsaná a zkompilovaná pro operační systém Linux, lze ji zkompilovat pod ostatními Unixovými systémy a na webových stránkách produktu jsou taktéž k dispozici zkompilované binární soubory pro systémy Windows. Podporuje čtení a zápis velkého množství formátů zvukových souborů.

Knihovna byla použita ve verzi 1.0.24, aktuální kde dni 16. 5. 2011. Dostupná je z oficiálních WWW stránek projektu http://www.mega-nerd.com/libsndfile/.

Pro zdárné zkompilování knihovny ve zdrojovém kódu je nutné ji z hlavičky souboru zavolat `#include <sndfile.h>` a při spuštění kompilátoru (v našem případě gcc) přidat do příkazového řádku parametr pro nalinkování knihovny `-lsndfile`.

3.2.1 Struktury a funkce knihovny *libsndfile*

Veškeré informace o načteném zvukovém souboru jsou uloženy ve struktuře `SF_INFO`, která je deklarovaná

```
typedef struct {
   sf_count_t frames;  //pocet vzorku
   int samplerate;     //vzorkovaci frekvence
   int channels;       //pocet kanalu
   int format;         //format souboru
   int sections;       //pocet sekci
   int seekable;       //moznost skakani
} SF_INFO ;
```

Pro účely implementace VAD jsou nejdůležitější složky `frames`, která uchovává hodnotu celkového počtu vzorků vstupního signálu z audio souboru, `samplerate` udávající vzorkovací frekvenci a `channels`, do které je uložen počet kanálů. V proměnné `format` je uložen formát zvukového souboru pomocí hexadecimální bitové masky. Knihovna *libsndfile* načítá formáty vstupních souborů automaticky z hlavičky souboru, proto není nutné při načítání zvukových souborů nic nastavovat. Konkrétní bitové masky pro jednotlivé formáty zvukových souborů jsou dostupné na oficiálních stránkách knihovny. Proměnné `sections` a `seekable` nejsou pro naše účely využity.

Funkce pro otevření zvukového souboru je deklarována

```
SNDFILE* sf_open (const char *path, int mode, SF_INFO *sfinfo);
```

kde `SNDFILE*` je anonymní ukazatel na data, která jsou privátní pro tuto knihovnu. Ukazatel `const char *path` odkazuje na název souboru, který má být načten, `SF_INFO *sfinfo` ukazuje na strukturu informací o zvukovém souboru popsanou výše. `int mode` se nastavuje podle způsobu zacházení se vstupním souborem:

- `SFM_READ` - soubor určen pouze ke čtení,
- `SFM_WRITE` - soubor určen pouze k zápisu,
- `SFM_RDWR` - soubor určen ke čtení i zápisu.

V dalším kroku je alokována paměť pro vzorky vstupního zvukového souboru

```
double *buffer = malloc(sfinfo.frames * sizeof(double));
```

a je provedeno jejich načtení vzorků do alokované paměti

```
sf_count_t sf_read_double
      (SNDFILE *sndfile, double *ptr, sf_count_t items);
```

Funkce vrací počet načtených vzorků, který lze pro ověření úspěšnosti porovnat s hodnotou `frames` ve struktuře `SF_INFO`. Ukazatel na proměnnou `double *ptr` ukazuje na alokované pole pro vzorky zvukového souboru a `sf_count_t items` udává hodnotu `frames` ve struktuře `SF_INFO`.

Pokud proběhne načtení zvukového souboru do paměti úspěšně, je ve struktuře `soubor` naplněna hodnota `buffer` adresou na datové pole s vzorky vstupního signálu a hodnota `success` je nastavena na 1. V opačném případě vypíše program chybovou hlášku.

3.3 Ruční značkování nahrávek řeči

3.3.1 Význam a provedení ručního značkování

Pro ověření funkčnosti implementovaného detektoru hlasové aktivity je výsledek detekce porovnán s ručně označkovanými testovanými soubory. Vzhledem ke možnostem lidského sluchu, kdy posluchač slyší, zda se v nahrávce právě vyskytuje řeč nebo pauza a zraku, kdy vidí daný zvukový soubor na monitoru, je možné naprosto přesně určit začátek a konec promluvy. Samozřejmě za podmínek, kdy jsou tyto jevy schopné lidské smysly zaregistrovat. Proto jsou tyto značky uváděny jako referenční při porovnání s počítačem označeným souborem.

Ruční značkování bylo provedeno v open-source programu Praat. V době realizace diplomové práce byla aplikace dostupná ve verzi 5.2.25 z oficiálních stránek http://www.fon.hum.uva.nl/praat/. Tato aplikace nabízí spoustu užitečných funkcí pro zpracování řeči. Jednou z nich je značkování hranic promluvy (zvané taktéž „lablování“).

Značky jsou uloženy v souboru **.TextGrid*. Ten je ke zvukovému souboru vytvořen pomocí volby *Annotate -> To TextGrid*. Zvukový soubor je spolu s tímto vytvořeným TextGrid souborem společné označen a zobrazen tlačítkem *View & Edit*. Značky jsou vytvářeny v nově zobrazeném okně buď kliknutím myši na příslušný řádek nebo pomocí menu *Boundary -> Add on selected tier*.

Program Praat nabízí množství výstupů souboru se značkami. Je možné vygenerovat seznamy, tabulky v textovém nebo binárním formátu. Pro naše účely je nutné vygenerovat výsledný **.TextGrid* soubor pomocí menu *Save -> Save as text file...*.

3.3.2 Načtení ručních značek do paměti

Pro načtení značek zapsaných v TextGrid souboru byla vytvořena funkce

```
int precpocet_rucni (struct vad_info *rucni_znacky, int fvz);
```

v souboru *pomocne_fce.c*. Vstupními parametry jsou ukazatel na strukturu obsahující informace o ručně značkovaném souboru, do které jsou následně zapsány hodnoty značky pro jednotlivé segmenty a vzorkovací frekvence. Výstupem je pouze informace o úspěšnosti provedené funkce hodnotou 1 nebo neúspěchu hodnotou 0.

Funkce nejprve otevře TextGrid soubor pomocí `fopen()`, vynechá ze souboru prvních jedenáct řádků, které jsou pro naše účely nepodstatné a načítá informace o začátku a konci souboru v sekundách a počet bodů, které byly při ručním značení označeny. Poté se body načtou a přepočítají pro jednotlivé segmenty a výsledek se uloží do pole `vad_info.vad`.

3.4 Výpočet detekce hlasové aktivity

Nyní se dostáváme k hlavní části zdrojového kódu. Výpočet detekce hlasové aktivity byl proveden pro oba typy detektorů popsané v teoretické části práce. Detektory hlasové aktivity jsou základním a většinou taky prvotním blokem v řetězci zpracování řečového signálu.

Přestože musí být detekce provedena s co největší spolehlivostí, požadujeme u tohoto bloku co nejkratší dobu zpracování obzvláště u systémů zpracování řeči v reálném čase. Je důležité, aby systém nechal dostatek času na zpracování dalších příznaků a ostatních operací zpracování řeči.

Tato kapitola se zaměřuje na možnosti implementace detektorů v jazyce C, porovnává rychlosti výpočtů a hodnotí volně dostupné matematické knihovny z hlediska jejich zátěže systému při výpočtech. Zvláštní kapitola je poté věnována způsobům měření procesorového času potřebného k výpočtu detekce.

3.4.1 Realizace pomocí knihovny *math.h*

Matematická knihovna *math.h* je standartně dostupná se základní instalací jazyka C. Po načtení v hlavičce zdrojového kódu příkazem

```
#include <math.h>
```

máme k dispozici více než dvacet nejčastěji používaných matematických funkcí spolu s důležitými konstantami.

Prvním realizovaným detektorem je energetický detektor. V přiložených zdrojových kódech je možné jej nalézt v souboru `vad.c` jako funkci

```
struct vad_info vad_energ_math (struct soubor *my_soubor,
                int L, int P, float Et_nula, float k2, float p2);
```

Nejprve je ve funkci naplněna struktura `vad_info` vstupními hodnotami (velikost segmentu L a přesahu P) a pomocí funkce `pocet_seg()` je vypočítán počet segmentů vstupního signálu. Na základě tohoto počtu je následně alokována paměť pro pole s hodnotami krátkodobé energie a prahové hodnoty počítané v jednotlivých segmentech.

Tyto hodnoty není nutné do paměti ukládat, algoritmus si vystačí s aktuálními hodnotami. Jejich ukládání do paměti bylo provedeno pro ověření správné funkčnosti algoritmu.

Další vstupní hodnoty jsou ukazatel na strukturu `struct soubor *my_soubor` s informacemi o načteném zvukovém souboru a pole s jeho vzorky. Hodnota `float Et_nula` je naplněna počáteční hodnotou prahu, který je dále v algoritmu přepočítáván podle rovnice 1.14. Do proměnné `float k2` je vložena hodnota, která se přičítá k vypočtené energii aktuálního segmentu při porovnání s prahem kvůli zamezení ovlivnění detekce náhodnými špičkami v signálu. Poslední vstupní hodnota `float p2` obsahuje koeficient vyhlazování, které se provádí v segmentech kde není detekována řeč.

Stěžejní částí algoritmu je výpočet energie v jednotlivých segmentech. Energii pročítáme ve `for` cyklu pro $(K-1)$ segmentů, protože u posledního segmentu není zaručeno, že bude mít právě délku jednoho segmentu L. Pokud by výpočet proběhl i pro poslední segment, který většinou nedosahuje požadované délky, sáhl by program mimo alokovanou paměť a jeho činnost by byla ukončena chybovým hlášením. Poslední segment je proto na závěr počítán zvlášť pouze pro počet vzorků, které do konce zvukového souboru zbývají.

Energie jednoho segmentu je počítána ve `for` cyklu s počtem iterací L. V každé iteraci je spočítána druhá mocnina aktuálního vzorku signálu a tato hodnota je přičtena k již vypočtené hodnotě předchozích vzorků daného segmentu. Po L opakování tak dostáváme výslednou energii daného segmentu.

Druhá mocnina je vypočtena pomocí funkce implementované v knihovně *math.h*

```
double pow (double base, double exponent);
```

Podle porovnání vypočtené energie E s prahem E_{T} navýšeným o hodnotu k_2 dostáváme rozhodnutí, zda je daný segment řeč nebo není. Výsledek se uloží do složky `vad[i]` struktury `vad_info`, kde proměnná i značí pořadí aktuálně zpracovávaného segmentu.

Pokud je segment vyhodnocen jako řečově neaktivní, proběhne ještě vyhlazení hodnoty aktuálního prahu podle rovnice 1.14. Po skončení výpočtu energie a vyhodnocení detekce je uvolněna paměť alokovaná pro hodnoty energie a prahu pomocí příkazu `free()`. Celá funkce vrací strukturu s informacemi a daty detekce `vad_info`.

Realizace energetického detektoru je taktéž provedena ve verzi bez alokace paměti pro hodnoty energie a prahu pro rozhodování. Tu je možné spustit voláním funkce

```
struct vad_info vad_energ_math_bezpole (struct soubor *my_soubor,
          int L, int P, float Et, float k2, float p2);
```

Funkce má stejné vstupní i výstupní parametry jako výše popsaná.

3.4.2 Realizace pomocí knihovny *gsl_ math.h*

GNU Scientific Library (GSL) je numerická knihovna pro programovací jazyky C a C++. Je volně šířena pod licencí GNU General Public Licence. Knihovna nabízí velké množství matematických funkcí pro vědecké účely s otevřeným zdrojovým kódem. Aktuální verze k 17. 5. 2011 použitá v této diplomové práci má označení GSL - 1.15. Knihovna je dostupná z oficiálních WWW stránek projektu http://www.gnu.org/software/gsl/.

Po instalaci knihovny z repozitáře dané Linuxové distribuce jsou automaticky nahrány potřebné hlavičkové soubory do složky, které kompilátor C zahrnuje do svých zahrnutých knihoven. Pokud je tedy knihovna standardně instalována není nutné provádět další dodatečné zahrnování složek.

V hlavičkovém souboru detektoru `vad.h` je mezi načítané knihovny nově zavedena matematická knihovna GSL pomocí příkazu

```
#include <gsl/gsl_math.h>
```

Při následné kompilaci je nutné přidat kompilátoru gcc parametr *-lgsl*.

Výpočet detekce hlasové aktivity probíhá stejným způsobem jako detektor popsaný v kap. 3.4.1. Rozdílné je ovšem provedení vlastního výpočtu energie v jednotlivých segmentech. Zde je využito funkce dostupné z knihovny GSL pro výpočet druhé mocniny

```
double gsl_pow_2 (const double x);
```

Knihovna byla realizována ve verzi s alokovanými poli pro uchování hodnot energie a prahu i bez nich, stejně jako funkce realizována pomocí standardní C knihovny *math.h*. Rychlost výpočtu bude porovnána v závěrečné části práce.

3.4.3 Zavedení struktury `gsl_vector`

Jednou z možností, které matematické knihovny GSL nabízejí pro interpretaci datových polí jsou vektorové struktury. Reprezentují části větších bloků a různé vektory můžou ukazovat na stejný blok. Tyto části jsou soubor po sobě jdoucích elementů paměti se stejnými mezerami mezi sebou.

Již zmiňované bloky (struktura `gsl_block`) jsou v hiearchii datových struktur knihoven GSL jakousi nadmnožinou struktury `gsl_vector`. Jednotlivé prvky vektoru jsou potom definovány jako prvky bloku s různými vzdálenostmi. Struktura `gsl_vector` je deklarována

```
typedef struct {
   size_t size;       //velikost vektoru
   size_t stride;     //vynechani hodnot
   double * data;     //ukazatel na pole s daty
   gsl_block * block; //ukazatel na blok
   int owner;         //opravneni k bloku
} gsl_vector;
```

Složka `size_t size` udává velikost vektoru, rozsah vektoru je tak dán intervalem $< 0; \text{size} - 1 >$. `size_t stride` definuje velikost skoku mezi jednotlivými elementy fyzické paměti, velikost skoku je dána v jednotkách odpovídajících datovému typu. Ukazatel na proměnnou typu `double * data` reprezentuje adresu prvního prvku vektoru ve fyzické paměti. Pokud jsou prvky elementu uloženy v bloku, ukazatel na něj lze nalézt jako složku `gsl_block * block`. Pokud vektor „vlastní“ tento blok, potom je proměnná `int owner` nastavena na hodnotu 1 a blok bude dealokován v momentě, kdy bude paměť vektoru uvolněna.

Knihovna s vektory se do programu zahrne pomocí načtení hlavičkového souboru

```
#include <gsl/gsl_vector.h>
```

Pro náš detektor hlasové aktivity bude využito dat, která jsou již načtena za pomocí funkcí knihovny *libsndfile* v poli `buffer`. Všeobecně pro čtení dat z polí za účelem jejich využití v rámci vektorů je definována funkce

```
gsl_vector_view gsl_vector_view_array (double * base, size_t n);
```

kde `double * base` je ukazatel na začátek fyzické paměti pole a `size_t n` je velikost pole, které má být načteno.

Návratová struktura `gsl_vector_view` je dočasný objekt uložený v zásobníku a může pracovat nad určitou podmnožinou prvků vektoru. Díky komponentě `vector` je s ním možné pracovat stejně jako s klasickým vektorem.

V dočasné struktuře `gsl_vector_view` máme načtené prvky jednoho segmentu a zbývá z nich vypočítat energii. Pro tento výpočet byla opět využita funkce knihovny GSL, tentokrát se však jedná o funkci knihoven BLAS (Basic Linear Algebra Subprograms), které GSL využívá a pouze poskytuje rozhraní k přístupu k nim.

Funkce knihovny BLAS jsou do projektu zahrnuty díky hlavičkovému souboru

```
#include <gsl/gsl_blas.h>
```

Z nich je potom využita funkce

```
double gsl_blas_dnrm2 (const gsl_vector * x);
```

Tato funkce počítá tzv. Euklidovskou normu podle rovnice

$$\|x\| = \sqrt{\sum_{i=1}^{L} x_i^2}. \tag{3.1}$$

Rovnice 3.1 je po umocnění na druhou rovna hledanému výsledku, který udává energii jednoho segmentu zpracovávaného signálu. Funkcí `gsl_pow_2()` se provádí umocnění, které byla popsána v kap. 3.4.2. Ostatní části v algoritmu jsou shodné s energetickým detektorem popsaným v kap. 3.4.1.

3.4.4 Realizace statistického detektoru

Pro implementaci statistického detektoru si nevystačíme se standardními prostředky jazyka C, protože podle rovnice 1.29 je k výpočtu poměru věrohodnosti zapotřebí rozptyl spektrálních komponent λ, které dostaneme výpočtem DFT (diskrétní Fourierovy transformace). Funkce, které knihovna pro tyto účely nabízí dostaneme zahrnutím hlavičkového souboru

```
#include <gsl/gsl_fft_complex.h>
```

Pro výpočet DFT využijeme funkci knihovny GSL pro výpočet dopředné rychlé Fourierovy transformace (FFT)

```
int gsl_fft_complex_forward (gsl_complex_packed_array data,
size_t stride, size_t n, const gsl_fft_complex_wavetable * wavetable,
gsl_fft_complex_workspace * work);
```

Vstupními parametry funkce jsou `gsl_complex_packed_array data`, která obsahuje komplexní data vstupního signálu. Do této proměnné jsou zároveň uloženy výsledky výpočtu FFT. Pole `data` musí být předem alokováno o délce rovné dvojnásobku velikosti segmentu. To je z důvodu, že do pole budou ukládány komplexní výsledky, tzn. s reálnou i imaginární složkou. Dále je vhodné nadefinovat rutiny, které budou pracovat zvlášť s komplexními i imaginárními komponentami

```
#define REAL(z,i) ((z)[2*(i)])
#define IMAG(z,i) ((z)[2*(i)+1])
```

Reálné složky proměnné `data` jsou následně naplněny vzorky vstupního zvukového souboru daného segmentu. Dalším parametrem funkce pro výpočet FFT je `size_t stride` definující velikost skoku mezi hodnotami vstupního pole. Pro naše účely je nastavena na hodnotu 1. Následuje velikost bloku dat `size_t n`. Protože není možné zaručit předem velikost segmentu o délce 2^n kvůli různé volbě délky segmentu a jeho přesahu, byla použita právě funkce `gsl_fft_complex_forward()`, která umožňuje vypočítat FFT o libovolné délce signálu.

Knihovna GSL obsahuje taktéž funkce pro výpočet FFT pro segmenty o délce 2^n, které jsou výpočetně snažší, ovšem podle referenční příručky algoritmus automaticky zvolí jednodušší verzi výpočtu při velikosti bloku 2^n. S tím souvisí i vnitřní funkce algoritmu, která pro signály o délce různé od 2^n využívá subtransformační moduly. Tyto moduly jsou vysoce optimalizované FFT malých délek, které jsou následně kombinovány pro vznik delší FFT [4].

Posledními vstupními parametry funkce pro výpočet FFT jsou `*wavetable` a `*work`. Ukazatel na proměnnou `*wavetable` musí být nejdříve alokován pomocí funkce

```
gsl_fft_complex_wavetable *
     gsl_fft_complex_wavetable_alloc (size_t n);
```

Touto alokací jsou připraveny vyhledávací trigonometrické tabulky. Délka `size_t n` je faktorizována na součin subtransformací a faktory a jejich trigonometrické koeficienty jsou uloženy přímo do tabulky `wavetable`. Trigonometrické koeficienty jsou počítány přímým voláním funkcí `sin()` a `cos()`. Kompletní struktura je k dispozici na webových stránkách knihovny GSL [4].

Alokaci paměti pro proměnnou `work` provedeme funkcí

```
gsl_fft_complex_workspace *
     gsl_fft_complex_workspace_alloc (size_t n)
```

Alokovaná struktura slouží pro průběžné ukládání výsledků při samotném výpočtu FFT. Po výpočtu FFT je vhodné uvolnit paměť alokovanou pro struktury `wavetable` a `work`.

Knihovny *gsl_ complex.h* a *gsl_ complex_ math.h* zajišťují práci s komplexními čísly. V realizovaném statistickém detektoru hlasové aktivity je využita funkce pro vyčítání dat z běžného datového pole

```
gsl_complex gsl_complex_rect (double x, double y);
```

a takto navrácená hodnota má datový typ `gsl_complex`, se kterým pracují všechny funkce nad komplexními čísly. Pro rovnici 1.29 je počítána druhá mocnina absolutní hodnoty z komplexního čísla funkcí

```
double gsl_complex_abs2(gsl_complex z);
```

Pro výpočet poměru věrohodnosti je ještě třeba funkce pro výpočet přirozeného logaritmu. Pro logaritmické výpočty obsahují matematické knihovny GSL speciální funkce, které zpřístupníme zahrnutím hlavičkového souboru

```
#include <gsl/gsl_sf_log.h>
```

Výpočet přirozeného logaritmu poté provedeme funkcí

```
double gsl_sf_log (double x);
```

V algoritmu je dále vyhodnoceno, zda poměr maximální věrohodnosti je větší než stanovený práh a na základě výsledku vyhodnoceno, zda se jedná o řeč nebo o segment bez řeči. Následně je přepočítána nová hodnota odhadu časově variantního spektra podle rovnice 1.35. Celá funkce pro statistickou detekci řečové aktivity je ukončena dealokací paměti potřebné pro výpočtu v rámci algoritmu a navrácením informací o detekci ve struktuře `vad_info`.

4 MĚŘENÍ PROCESOROVÉHO ČASU

4.1 Definice procesorového času

Pod pojmem „čas“ rozumíme jeden konkrétní bod v souvislém časovém kontinuu, např. 8. června 2011 10:00 hod. SELČ (středoevropského letního času). Intervalem rozumíme souvislou část časového kontinua mezi dvěma body v kalendářním čase, např. 10.-12. února 2011. Uplynulý čas je délka intervalu, např. 25 minut. Časový rozsah je suma uplynulého času, která nemusí být součástí nějakého konkrétního časového intervalu, např. vývoj programu trval 30 hodin (hodnota je součtem více intervalů v různém čase).

Doba trvání je uplynulý čas intervalu mezi dvěma událostmi. Můžeme ji také nazvat periodou pokud se jedná o součást pravidelně opakovaného cyklu. Procesorový čas je jako čas v kalendáři s tím rozdílem, že je založen na podmnožině časového kontinua kdy určitý proces využívá procesor. Procesorový čas je tedy relativní k procesu a určuje součet časů kdy je procesor využíván [4].

Program využívá CPU instrukcemi procesu a procesorovým časem se rozumí doba tohoto využití. Není do něj započítán čas kdy se např. čeká na vstupní/výstupní operace. Je měřen v hodinových taktech nebo jako procento CPU kapacity a počítá se od určité události (v GNU je to vytvoření procesu). Procesorový čas jednoho procesu tak můžeme stanovit tak, že změříme čas dvou událostí a nižší hodnotu odečteme od vyšší. U jednoprocesorových systémů je procesorový čas buď stejný a nebo menší než čas reálný.

U multiprocesorových strojů, které využívají paralelní zpracování procesů je procesorový čas součtem všech časů, které jednotlivé CPU jednotky potřebují k zpracování procesu. U těchto systémů je tedy procesorový čas delší než reálný

Reálný čas (tzv. „Wall Clock“), který se počítá od startu počítače až po jeho vypnutí je měřen klasickými hodinami. Zahrnuje proto i všechny časy vzniklé čekáním programu. Pro účely měření procesorového času není vhodné používat reálný čas, protože procesor může čekat na vstup/výstup programu a nemusí při tom pracovat [2].

4.2 Standardní prostředky jazyka C pro měření procesorového času

4.2.1 Struktury a funkce knihovny *time.h*

Jazyk C nabízí standardně knihovnu, jejíž hlavičkový soubor *time.h* deklaruje typy a funkce pro manipulaci s daty a časem. Obsahuje funkce, které pracují s lokálním časem, jenž se může lišit od kalendářního času vzhledem k časovému pásmu nebo letnímu a zimnímu času. V knihovně jsou definovány aritmetické typy `clock_t` a `time_t`, které reprezentují čas. Pro měření procesorového času se používá první z nich a její ekvivalentní datový typ je `long int`. Nutné je ovšem dát pozor na implementaci v různých systémech, protože `clock_t` může být interpretováno v různém rozlišení a přesnosti [7].

Nejjednodušší způsob jak vypočítat procesorový čas je sejmout hodnotu `clock_t` pomocí funkce `clock()` před začátkem měřeného procesu a po jeho provedení a výsledné hodnoty odečíst:

```
clock_t start, konec;   //hodnoty CPU time
double vysledek;        //vysledny cas

start = clock();        //zacatek mereni casu
/* Mereny proces */
konec = clock();        //konec mereni casu
vysledek = ((double) (konec - start)) / CLOCKS_PER_SEC;  //vysledek
```

Konstanta `CLOCKS_PER_SEC` udává počet hodinových taktů za sekundu. Na 32-bitových systémech je definován v hodnotě jeden milion, ale stejně jako `clock_t` může mít v různých systémech různou hodnotu. Výsledek je v sekundách, ale je zaokrouhlen na jednotky, proto není dostatečně přesnou metodou pro měření procesorového času.

Zřejmě nejpřesnější metodou pro měření procesorového času je možné díky další struktuře knihovny *time.h* a s ní souvisejícími funkcemi. Struktura `timespec` je deklarována

```
struct timespec {
   time_t   tv_sec;    //pocet sekund
   long     tv_nsec;   //pocet nanosekund
};
```

Složka `tv_sec` určuje čas v sekundách a `tv_nsec` udává čas v nanosekundách. Poskytuje tak velmi dobrou přesnost pro měření procesorového času. Funkce, která umožňuje získat aktuální hodnotu času je

```
int clock_gettime(clockid_t clk_id, struct timespec *tp);
```

Vstupními parametry funkce jsou ukazatel na strukturu `struct timespec *tp`, do které je sejmutý čas uložen a argument `clockid_t clk_id`, který slouží jako identifikátor specifických hodin, ze kterých je čas odečítán. Tyto hodiny můžou být v rámci celého systému a viditelné pro všechny procesy, nebo můžou měřit čas v rámci jednoho samostatného procesu. Identifikátor `clk_id` může v současné verzi GNU libc na Linuxovém jádře nabývat hodnot:

- **CLOCK_REALTIME** - Hodiny počítající reálný čas v rámci celého systému. Nastavení tohoto počitadla je možné pouze s příslušným oprávněním a jako jediné je spolehlivě použitelné na všech platformách. Jediné hodiny, které můžou být nastaveny pomocí funkce `clock_settime()`,
- **CLOCK_MONOTONIC** - Hodiny které nemůžou být nastaveny. Reprezentují monotonní čas od nějaké nespecifikované počáteční události,
- **CLOCK_PROCESS_CPUTIME_ID** - Hodiny pro individuální procesy ve vysokém rozlišení,
- **CLOCK_THREAD_CPUTIME_ID** - Totéž jako předchozí, měří čas jednotlivých vláken.

Měření je prováděno stejně jako v předchozím případě, funkcí `clock_gettime()` je sejmuta hodnota před a po provedení měřeného procesu a hodnoty jsou poté odečteny. V našem případě byl pro sejmutí aktuálního času použit identifikátor `CLOCK_PROCESS_CPUTIME_ID`, protože poskytuje nejvyšší přesnost a měření pro specifické procesy.

Z důvodu odděleného vyjádření celé a desetinné části času ve struktuře `timespec` je nutné provádět odečítání jednotlivých sejmutých časových hodnot pomocí speciální funkce. Tato funkce byla v programu definována následovně:

```
struct timespec rozdil(struct timespec start,struct timespec end){
   struct timespec temp;
   if((end.tv_nsec - start.tv_nsec)<0){
      temp.tv_sec = end.tv_sec-start.tv_sec -1;
      temp.tv_nsec = 1000000000 + end.tv_nsec - start.tv_nsec;
   } else {
      temp.tv_sec = end.tv_sec - start.tv_sec;
      temp.tv_nsec = end.tv_nsec - start.tv_nsec;
   }
   return temp;
}
```

Její podstata spočívá v tom, že pokud je čas v nanosekundách odečtený po skončení procesu menší, než počet nanosekund před začátkem procesu, mohlo by při běžném

odečtení hodnot dojít k chybě výpočtu v celé části, proto je tato situace ve výše uvedené funkci ošetřena. Funkce vrací v hodnotu v sekundách, pro lepší orientaci ve výsledcích je čas přepočítán do řádu milisekund a uložen do hodnoty struktury `vad_info.cas`

```
info.cas_timespec = (double)vysl.tv_sec*1000;
info.cas_timespec += (double)vysl.tv_nsec*0.000001;
```

4.2.2 Struktury a funkce knihovny *sys/time.h*

Tato knihovna poskytuje prostředky pro měření reálného kalendářního času. Pro měření přesného výpočetního času procesů proto není zcela vhodná, ovšem její rozlišení dosahuje vyšší přesnosti než měření běžného reálného času. Ten je reprezentován v řádu sekund, zatímco knihovna *sys/time.h* nabízí měření reálného času v mikrosekundách. Proto byl v programu pro detekci hlasové aktivity tento způsob měření implementován a jeho výsledky porovnány s měřením procesorového času. Do paramtetrů komplilátoru je nutné přidat příkaz *-lrt*.

Základem je struktura `timeval` definovaná podobně jako struktura `timespec`:

```
struct timeval {
   long int   tv_sec;   //pocet sekund
   long int   tv_usec;  //pocet mikrosekund
};
```

s tím rozdílem, že složka `tv_usec` obsahuje čas v řádu milisekund. Funkce pro sejmutí aktuálního času je má deklaraci

```
int gettimeofday (struct timeval *tp, struct timezone *tzp);
```

Skrze ukazatele na strukturu `struct timeval *tp` je uložena aktuální hodnota reálného času a do struktury `tzp` jejíž adresu definuje ukazatel `struct timezone *tzp` je vrácena informace o časové zóně. Pokud je argument `tzp` NULL ukazatel, informace o časové zóně je ignorována. Výpočet rozdílu časů probíhá stejným způsobem, jako u hodnot struktury `timespec` pouze s jinou konstantou při přepočtu mikrosekund

```
struct timeval rozdil_tv(struct timeval start,struct timeval end){
   struct timeval temp;
   if((end.tv_usec - start.tv_usec)<0){ //
      temp.tv_sec = end.tv_sec-start.tv_sec -1;
      temp.tv_usec = 1000000 + end.tv_usec - start.tv_usec;
   } else {
```

```
        temp.tv_sec = end.tv_sec - start.tv_sec;
        temp.tv_usec = end.tv_usec - start.tv_usec;
    }
    return temp;
}
```

a výsledný čas je uložen do proměnné typu `double` pomocí přepočtu na milisekundy

```
info.cas_timeval  = (double)tv_vysl.tv_sec * 1000;
info.cas_timeval += (double)tv_vysl.tv_usec * 0.001;
```

4.3 Měření v prostředí Octave

Oproti zadání diplomové práce, které spočívalo v porovnání rychlosti výpočtu detekce hlasové aktivity v jazyce C a v prostředí Matlab bylo pro toto porovnání zvoleno prostředí Octave, které je volně dostupnou open-source alternativou. Octave bylo zvoleno pro dosažení co největší shody testovacích podmínek v operačním systému Linux. Algoritmus, jenž je popsaný a realizovaný v kap. 2.2, byl použit pro měření v prostředí Octave s drobnými úpravami. Jeho zdrojový kód je přístupný na přiloženém CD.

Pro měření procesorového času v prostředí Octave se nabízí funkce `cputime`, která vrací hodnotu aktuálního času CPU od doby, kdy byla naše aplikace spuštěna. Tatáž funkce se nachází i v prostředí Matlab a její funkčnost je naprosto stejná. Jednotkou vrácené hodnoty jsou sekundy.

4.4 Profilování aplikací

Jedním z témat, které má s měřením procesorového času hodně společného je profilování. Jedná se o statistickou analýzu aplikace a jejích jednotlivých funkcí. Výsledkem je ohodnocení jednotlivých částí kódu podle času, který je potřebný k jeho výpočtu, podle paměťové náročnosti atd. Jednotlivé prostředky, často volně dostupné ke stažení ve formě open-source, nabízí velké množství možností, které parametry v daném programu sledovat.

Aplikace, která profilování zajišťuje se nazývá Profiler. Všeobecně existují dvě hlavní fáze profilování:

1. Měření - získávání statistických údajů o běžícím programu. Jsou zaznamenávány především tyto hodnoty:
   - Počty volání jednotlivých funkcí,
   - počty iterací v jednotlivých cyklech,

- počty přístupů do paměti, na disk,
- doba strávená při těchto činnostech.

2. Profilovací analýza - získávání statistických informací z naměřených hodnot. Dle možností konkrétního profileru je možné sledovat hodnoty v tabulkách, grafech a analyzovat výsledky [10].

4.4.1 Profilování pomocí nástroje Valgrind

Valgrind je sada nástrojů pro tvorbu dynamických analytických pomůcek. Je vhodný především pro sledování správy paměti a procesorových vláken a poskytuje velmi přesné výsledky. Je volně dostupný ve formě open-source z oficiálních webových stránek http://valgrind.org/ spolu s veškerou potřebnou dokumentací. V této práci byla použita verze 3.5.0.

Pro naše účely byla využita verze, která nástroje Valgrind implementuje přímo do vývojového prostředí Eclipse. Profilování je možné vyvolat jednoduše pomocí kliknutí pravým tlačítkem na název projektu v Project Exploreru a následně vybrat možnost *Profile As -> Profile With Valgrind*. Proběhne profilování s implicitním nastavením stejně jako bychom spouštěli Valgrind z příkazové řádky bez parametrů. Výsledkem je zobrazení v okně Valgrind chyby ve správě paměti, např. pokud neuvolníme alokovanou paměť. Řádky, ve kterých se nedostatky nachází jsou označeny standardním chybovým zabarvením prostředí Eclipse.

V menu můžeme volbou *Run -> Profile As -> Profile Configuration* zvolit možnosti profilování. Vybereme kartu *Valgrind Options* a změníme hned první volbu *Tool to run* z *memcheck* na *massif* a volby uložíme. Dostaneme tak možnost analyzovat datové úložiště. Po profilování naší aplikace s touto konfigurací profileru můžeme vidět přehledný graf o využití paměti aplikací. Jednotlivé naměřené body se nazývají snapshots a každý z nich má své pořadí, CPU čas, ve kterém byl proveden, počet alokovaných bajtů.

Ve výsledcích této diplomové práce jsou grafy uvedeny spolu s porovnáním množství využité paměti pro každý z detektorů hlasové aktivity.

4.4.2 Profilování pomocí nástroje OProfile

OProfile je profilovací nástroj pro operační systém Linux, který nabízí profilování s minimálním zatížením systému. Jeho součásti tvoří ovladač pro kernel, program na pozadí pro sbírání potřebných dat a několik nástrojů pro zpracování sebraných výsledků. Využívá čítače času CPU. Přestože se jedná vývojově o alfa verzi, program je stabilní na většině dostupných systémech.

Verze použitá v projektu má označení 0.9.6 a je volně dostupná ke stažení z repozitáře konkrétní Linuxové distribuce nebo z oficiálních webových stránek programu http://oprofile.sourceforge.net/. Program je stejně jako Valgrind open-source. Taktéž je možné využít doplněk přímo do vývojového prostředí Eclipse.

Profilování pomocí OProfile spustíme kliknutím pravým tlačítkem na název projektu v Project Exploreru a následně vybrat možnost *Profile As -> Profile With OProfile*. Výsledky profilování se zobrazí v okně OProfile. Podle zvoleného parametru, jehož hodnoty se v průběhu měření zaznamenávají, můžeme pozorovat výsledky.

Pro posouzení výkonu našeho programu je využito implicitního nastavení parametru `CPU_CLK_UNHALTED`. Jedná se o událost, která počítá hodinové cykly procesoru kdy pracuje a není pozastaven. Název `CPU_CLK_UNHALTED` je společný pro většinu procesorů od společnosti Intel, u jiných výrobců může mít tato událost jiný název, všechny dostupné události s popisem jsou k dispozici na oficiálních stránkach OProfileru.

Výstupem profileru s takovým nastavením je procentuální rozložení výpočetního času v rámci celého programu. Na jednotlivých řádcích výstupu můžeme vidět funkce (popsané informacemi z debuggeru) s konkrétními hodnotami kolik uvedená funkce (resp. řádek ve zdrojovém kódu) v procentech zabral výpočetního výkonu z celkového výpočetního času aplikace.

Profilerem OProfile byly měřeny na výpočetní čas oba typy detektorů hlasové aktivity a výsledky jsou vyobrazeny do grafů a porovnány v závěrečné části diplomové práce.

5 VÝSLEDKY MĚŘENÍ

Podle výše uvedených poznatků o procesorovém času byly implementované detektory hlasové aktivity změřeny a testovány na rozložení výpočetního času profilery Valgrind a OProfile. Měření probíhalo na notebooku značky Hewlett-Packard ProBook 4510s s procesorem Intel®Core™2 Duo CPU T5870 (taktovací frekvence 2, 00 GHz).

5.1 Výsledky měření procesorového času

Měření procesorového času bylo provedeno pomocí prostředků knihoven *time.h* a *sys/time.h* popsaných v kap. 4.2.1 a 4.2.2. Pro každou implementaci bylo provedeno 101 měření procesorového času. Měření vykazovala náhodné chyby, které se projevovaly zvýšením procesorového času výpočtu až trojnásobně. Pro 10 měření se jednalo průměrně o dvě až tři hodnoty, které byly touto náhodnou chybou postiženy. Zbylé hodnoty už vykazovaly minimální odchylku. Proto byl zvolen jako výstupní prvek měření medián (z toho důvodu byl taky zvolen lichý počet měření). K výsledku byla taktéž vypočítána směrodatná odchylka. Hodnoty procesorového času jednotlivých realizací energetického detektoru jsou v tab. 5.1.

Algoritmy byly testovány pro nahrávku o délce přibližně 6 s, $f_{\mathrm{vz}} = 8000\,\mathrm{Hz}$. Přesná délka nahrávky je 47 335 vzorků. Tatáž nahrávka byla použita pro testování algoritmů realizovaných v prostředí Matlab a popsaných v kap. 2.

Poznámka „s polem“ označuje realizace, při kterých byly v paměti uchovávány všechny změřené hodnoty energie segmentu E a prahu pro rozhodování E_{T}. Tyto hodnoty není nutné uchovávat, ukládání do pole bylo zavedeno z důvodu ověření funkcí algoritmu a jak je z naměřených hodnot vidět, zvyšuje výpočetní čas algoritmu až dvojnásobně.

Implementace, které toto pole neuchovávají nepřekročí délku výpočtu 1 ms. Výpočet realizovaný standardní knihovnou jazyka C *math.h* je přibližně o 0, 1 ms rychlejší než realizace pomocí knihovny *gsl_math.h*. V takto krátkých souborech není

Knihovna, pozn.	t_{timespec} [ms]	$\sigma_{\mathrm{timespec}}$ [ms]	t_{timeval} [ms]	$\sigma_{\mathrm{timeval}}$ [ms]
math.h, bez pole	0, 6418	0, 0225	0, 6470	0, 0202
math.h, s polem	1, 0617	0, 0061	1, 1380	0, 0689
gsl_math.h, bez pole	0, 7630	0, 0037	0, 7850	0, 0457
gsl_math.h, s polem	1, 3181	0, 0058	1, 3090	0, 0077
gsl_vector.h	1, 5412	0, 0034	1, 5330	0, 0080

Tab. 5.1: Procesorový čas energetického detektoru

tento rozdíl nějak podstatný, u zpracování delších signálů nebo u aplikací pracujících v reálném čase může být každá úspora výpočetního času ku prospěchu, zvláště pokud následují další bloky zpracování jiných vlastností řeči a potřebují co nejrychlejší výsledky o detekci hlasové aktivity.

Využitím knihovny *gsl_ vector.h* pro uchovávání jednotlivých segmentů ve struktuře `gsl_vector_view` bylo dosaženo jednodušší práce se segmenty díky funkcím, které knihovna nabízí, což ovšem mělo za následek zvýšení procesorového času této implementace. Výsledné hodnoty jsou přibližně $2,5$krát větší, než u nejrychlejší realizace pomocí standardní knihovny *math.h*. Toto navýšení je způsobeno převodem informací do struktur knihovny *gsl_ vector.h*, což může být u takto jednoúčelové a jednoduše realizovatelné funkce jako je energetický detektor hlasové aktivity zbytečné. Knihovna však určitě najde své uplatnění při větších a rozsáhlejších projektech.

Co se týče porovnání knihoven pro měření procesorového času, jejich výsledky jsou téměř shodné. Nevykazují větší rozdíly ve výsledcích než $0,07$ ms, což je zanedbatelná hodnota. Výhodou může být využití struktury `timespec` díky její větší přesnosti. Směrodatné odchylky pro 101 provedených měření dosahují minimálních hodnot, pouze u první realizace struktury `timespec` a prvních třech realizací struktury `timeval` dosahují odchylky rádu setin milisekund, což je o jeden řád více, než u ostatních realizací a měření. Jedná se zřejmě o náhodné chyby v měření.

Statistický detektor byl taktéž podroben měření procesorového času potřebného pro výpočet detekce hlasové aktivity. Stejně jako u energetického detektoru bylo provedeno 101 měření a vybrán medián. Jak je vidět v tab. 5.2 je z výsledků jasně patrný nárust času, který je pro detekci hlasové aktivity nutný. Oproti nejrychlejšímu energetickému detektoru se zvýšil přibližně třicetinásobně do řádu desítek milisekund. Nejnáročnější operací v tomto algoritmu, který procesorový čas zvýšila, je výpočet FFT. Statistický detektor však umožňuje daleko lepší výsledky detekce hlasové aktivity, jak je zřejmé z výsledků v kap. 2.2. Jejich využití může být vhodnější vzhledem k výsledkům a úspěšnosti detekce v aplikacích, které nepracují v reálném čase nebo v realtimových aplikacích, pro něž není větší zpoždění překážkou.

Při srovnání výsledků s výpočtem v prostředí Octave vidíme, že jazyk C je o jeden celý řád rychlejší. Důvodem k delšímu výpočtu v prostředí Octave může být jeho vnitřní režie pro zpracování algoritmu.

t_{timespec} [ms]	σ_{timespec} [ms]	t_{timeval} [ms]	σ_{timeval} [ms]	t_{octave} [ms]	σ_{octave} [ms]
$19,7219$	$0,5440$	$19,8140$	$0,6415$	$131,98$	$0,0013$

Tab. 5.2: Procesorový čas statistického detektoru

5.2 Měření rozložení výpočetních prostředků pomocí OProfile

Za pomoci aplikace bylo provedeno měření, jehož výsledkem je procentuální rozložení procesorového času v rámci algoritmů pro výpočet detekce hlasové aktivity. Výsledky měření jsou uvedeny v následujících tabulkách.

V tab. 5.3 vidíme výsledky profilování energetického detektoru hlasové aktivity. Konkrétní výsledky jsou pro detektor realizovaný knihovnou *math.h*, jehož výpočet byl vyhodnocen jako nejrychlejší.

Největší procesorový čas zabírá výpočet funkce druhé mocniny `pow()` a to přibližně 87,5 % z celkového času výpočtu energetického detektoru. Dále, asi 11,7 % času trvá operace `for` cyklu nad výpočtem druhé mocniny. Zbytek, necelé 1 %, zabere zbytek příkazů v algoritmu.

Poměr doby výpočtu [%]	Funkce
87,50 %	`pow()`
11,67 %	`for` cyklus nad funkcí `pow()`
0,83 %	ostatní funkce

Tab. 5.3: Procesorový čas v jednotlivých fázích algoritmu energetického detektoru

Velmi podobné výsledky dosahovaly i ostatní algoritmy energetické detekce hlasové aktivity. Z výsledků je patrné, že pokud je někde možné dosáhnout úspory procesorového času ve výpočtu energetické detekce hlasové aktivity, je to právě blok výpočtu druhé mocniny. Jedním z předpokladů, jak tento výpočet zoptimalizovat, bylo zavedení matematických knihoven GSL, které by měly být optimalizované pro numerické výpočty. Z výsledků, které byly prezentovány v kap. 5.1 je zřejmé, že k žádnému zlepšení nedošlo.

U statického detektoru už je patrné rozložení výpočetního času do více matematických funkcí, které jsou pro jeho výpočet nutné. Výsledky tohoto měření jsou v tab. 5.4 a vidíme na nich, že nejvíce času, asi 30 %, zabírá výpočet poměru maximální věrohodnosti Λ, následuje výpočet rozptylu spektrálních komponent λ a hranici 20 % ještě překračuje `for` cyklus nad operací výpočtu Λ. Poslední pětinu výpočetního času tvoří ostatní funkce v algoritmu.

5.3 Měření využití paměti pomocí Valgrind

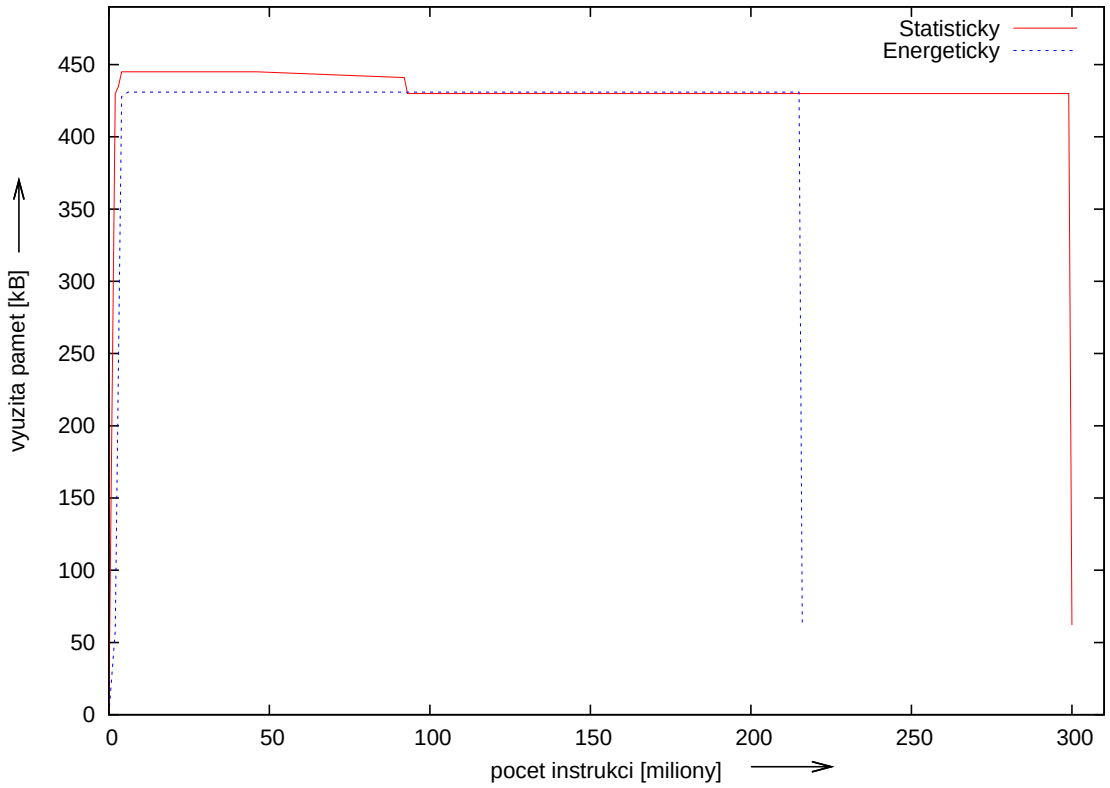
Paměťová náročnost detektorů řečové aktivity byla jako prostředek posouzení algoritmů zvolena z důvodu ohodnocení jednotlivých realizací z hlediska využití paměti.

Poměr doby výpočtu [%]	Funkce
$30,32\,\%$	výpočet Λ
$26,52\,\%$	výpočet λ
$22,10\,\%$	`for` cyklus nad výpočtem Λ
$6,63\,\%$	výpočet $\lvert X\rvert^2$, řádek 73
$6,08\,\%$	výpočet $\lvert X\rvert^2$, řádek 98
$8,35\,\%$	ostatní funkce

Tab. 5.4: Procesorový čas v jednotlivých fázích algoritmu statistického detektoru

Soubor nástrojů Valgrind a způsob jeho měření využití paměti byl popsán v kap. 4.4.1.

V grafu na obr. 5.1 je vidět, že využití paměťových prostředků je u obou realizací detektoru přibližně stejné. V průběhu prvních pěti milionů instrukcí naroste skokově velikost využité paměti na asi 440 kB především z důvodu alokace nejnutnějších proměnných a pole pro vzorky zvukového souboru. Tato paměť je prakticky nezměněna alokována až do konce výpočtu detekce. Algoritmus končí svoji činnost po asi 220



Obr. 5.1: Paměťová náročnost algoritmů detektorů.

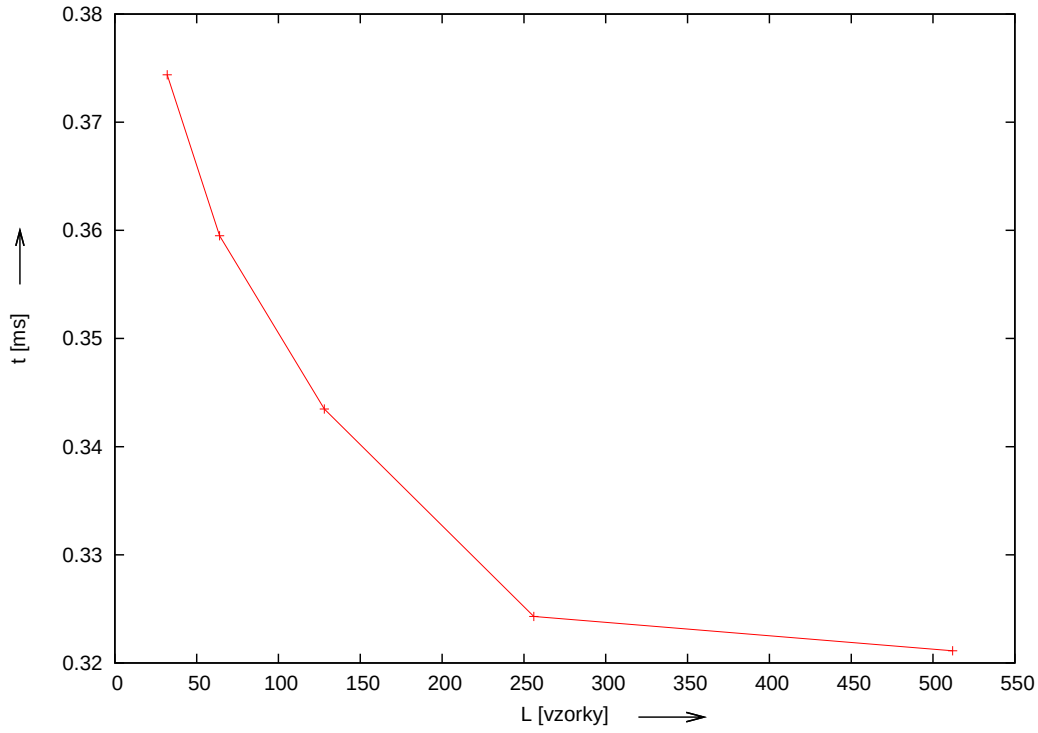
milionech provedených instrukcí a alokovaná paměť je uvolněna.

Statistický detektor využívá pro svůj výpočet několik málo alokovaných polí pro komplexní hodnoty spektra, proto v první části grafu je jeho použitá paměť asi o 10 kB vyšší. V konečném důsledku ale není o mnoho paměťově náročnější než energetický detektor.

Profilování pomocí Valgrindu bylo provedeno pro kompletní zdrojový kód realizace detektorů, ale samotná detekce je ve skutečnosti kratší. Dlouhý úsek viditelný v grafu mezi 100 miliony až 300 miliony instrukcí je dán především přepočtem hodnot VAD na reálné vzorky a jejich výpisem do textového souboru.

5.4 Délka procesorového času v závislosti na velikosti segmentu

Posledním testovaným jevem je závislost délky procesorového času na velikosti segmentu zpracovávaného detektorem. Pro porovnání výsledků je nutné si uvědomit, že čím kratší velikost segmentu je zvolena, tím větší je počet zpracovávaných segmentů



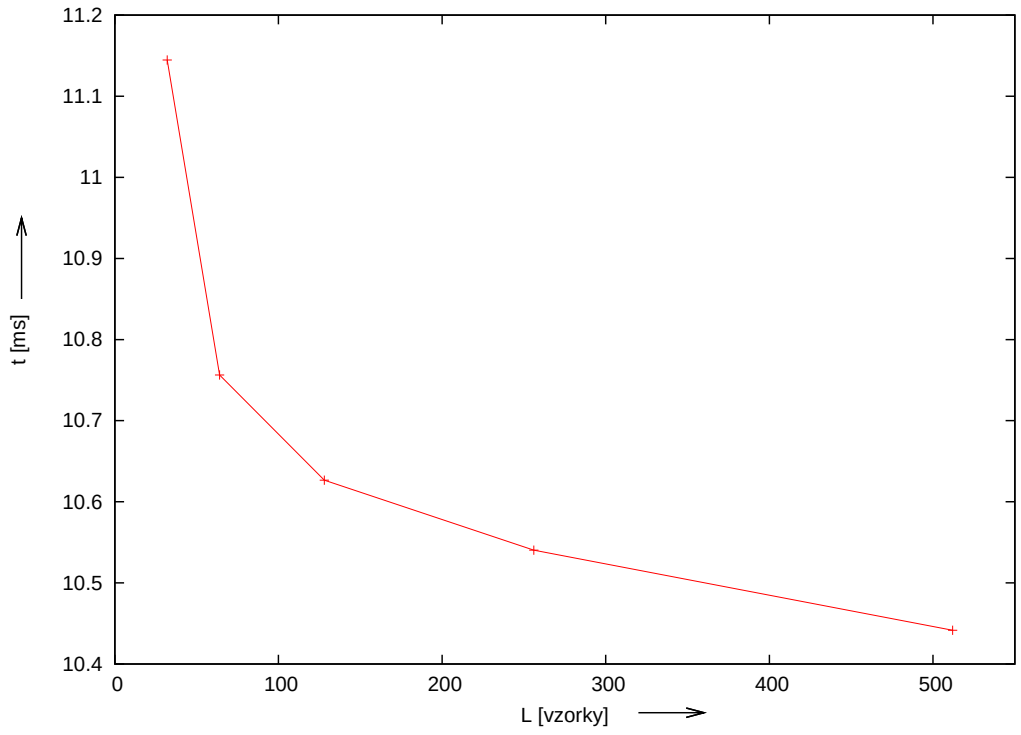
Obr. 5.2: Procesorový čas výpočtu energetického detektoru v závislosti na velikosti segmentu.

a tím pádem i počet iterací výpočtu příznaku pro detekci hlasové aktivity. Pro testovací účely byla zvolena stejná nahrávka jako v předchozích případech testování na procesorový čas a paměťovou náročnost.

Přesah jednotlivých segmentů byl kvůli objektivitě nastaven na 0. Testování bylo prováděno pro délky segmentů 32, 64, 128, 256 a 512 vzorků. Výsledky je možné vidět v následujících grafech.

Na obr. 5.2 můžeme pozorovat, že délka procesorového času se stoupající velikostí segmentu klesá, což je dáno klesajícím počtem iterací výpočtu energie. Zlom nastává u segmentu délky 256 vzorků, kde se průběh výrazně mění a vidíme už minimální rozdíl mezi rychlostí výpočtu algoritmu energetického detektoru s délkou segmentu 256 a 512 vzorků. Obr. 5.3 ukazuje výsledky statistického detektoru, které jsou velmi podobné výsledkům detektoru energetického s tím rozdílem, že celková doba výpočtu je pochopitelně několikanásobně větší.

U obou grafů vidíme, že výpočet je nejrychlejší pro segmenty délky 512 vzorků, ovšem na úkor úspěšnosti detekce. Ta nedosahuje nejlepších výsledků pro velmi malé (32 vzorků) nebo velmi velké (512 vzorků) hodnoty. Její volba je závislá na konkrétní aplikaci, úspěšnosti detekce, potřebné rychlosti zpracování atp.



Obr. 5.3: Procesorový čas výpočtu statistického detektoru v závislosti na velikosti segmentu.

6 ZÁVĚR

V rámci této diplomové práce byla nastudována problematika detekce hlasové aktivity. Pro realizaci byly zvoleny detektory na základě posouzení energie signálu a statistických vlastností spektra signálu. Obě tyto metody jsou teoreticky popsány s jejich matematickým základem v první části práce. Obě metody byly ověřeny v prostředí Matlab a z jejich výsledků je patrné, že statistický detektor dosahuje lepších výsledků detekce, než detektor energetický.

Detektory byly prověřeny na testovací databázi nahrávek, která byla pro účely této práce zhotovena. Pro nahrávky, kde průměrný odstup užitečného signálu od šumu činí 6 dB dosahuje statistický detektor úspěšně detekovanou řeč v 89, 3 % nahrávky a chybně detekovanou řeč ve 14, 8 %.

Detektory byly poté realizovány v programovacím jazyce C v operačním systému Linux za použití open-source knihoven. Nejprve byla provedena realizace energetického detektoru za pomoci standardních knihoven jazyka C, konkrétně *math.h* pro nahrávku v délce 6 s a $f_{\mathrm{vz}} = 8000\,\mathrm{Hz}$, délkou segmentu 256 vzorků a přesahem 128 vzorků trval výpočet detekce 0, 6418 ms. Tentýž soubor byl dále testován pro výpočet pomocí knihovny GSL *gsl_ math.h* s dobou výpočtu 0, 7630 ms a nakonec pomocí knihovny *gsl_ math.h* s dobou výpočtu 1, 5412 ms. Z výsledků je patrné, že nejrychlejší výsledky podává pro tento typ detektoru standardní C knihovna *math.h*.

Dále byl zrealizován statistický detektor hlasové aktivity rovněž pomocí knihoven GSL, zejména pro výpočet FFT. Jeho doba výpočtu 19, 7219 ms byla přibližně třicetinásobně pomalejší než nejrychlejší energetický detektor, umožňuje však daleko přesnější detekci. Ve srovnání se stejným detektorem realizovaným v prostředí Octave, jehož výpočet trval 131, 98 ms vychází procesorový čas jednoznačně ve prospěch realizace v jazyce C. Pomalejší doba výpočtu je způsobena vnitřní režií výpočtů v prostředí Octave.

Při porovnání paměťové náročnosti obou algoritmů bylo dosaženo podobných výsledků. Statistický detektor potřebuje při svém výpočtu přibližně o 5 % více paměti než detektor energetický.

Jednotlivé fáze výpočtu byly porovnány profilováním pomocí nástroje OProfile. V algoritmu energetického detektoru byl časově nejnáročnější proces výpočtu druhé mocniny, který zabral asi 87 % celkového výpočtu. U statistického detektoru to byly funkce pro výpočet poměru maximální věrohodnosti a rozptylu spektrálních složek, jejichž výpočet trval asi 78 % celkového procesorového času.

Při porovnání rychlosti výpočtu pro jednotlivé délky segmentu byly oba algoritmy nejpomalejší pro segment délky 32 vzorků a naopak nejrychlejší pro segment délky 512 vzorků. Počet segmentů je totiž nepřímo úměrný jejich délce.

LITERATURA

[1] BAŠTINEC, Jaromír. *Pravděpodobnost, statistika, operační výzkum.* Brno, 2009. 130 s. [skriptum]

[2] CPU time. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg(Florida): Wikipedia Foundation, 18. července 2006, last modified on 26. dubna 2011 [cit. 2011-05-18]. Dostupné z WWW: <http://en.wikipedia.org/wiki/CPU_time>.

[3] EPHRAIM, Yariv; MALAH, David Speech Enhancement Using a Minimum-Mean Square Error Short-Time Spectral Amplitude Estimator. In *Acoustics, Speech and Signal Processing, IEEE Transactions.* Prosinec 1984. s. 1109 - 1121. ISSN 0096-3518.

[4] *GNU Scientific Library Homepage* [online]. květen 2011 [cit. 2011-05-17]. Reference Manual. Dostupné z WWW: <http://www.gnu.org/software/gsl/manual/html_node/>.

[5] CHANG, Joon-Hyuk; KIM, Nam Soo; MITRA, Sanjit K. Voice Activity Detection Based on Multiple Statistical Models. In *Signal Processing, IEEE Transactions.* Červen 2006. s. 1965 - 1976. ISSN 1053-587X.

[6] KAČUR, Juraj The Concept of Task Specific Speech Database for VAD Systems. In *48th International Symposium ELMAR.* Zadar, Chorvatsko, Červen 2006. s. 155-158. Dostupné z WWW: <http://www.ktl.elf.stuba.sk/k̃acur/clanky/ConceptOfTaskSpecificSpeechDatabaseForVADSystems_zadar2006 .pdf>.

[7] KERNIGHAN, Brian W.; RITCHIE, Dennis M. *Programovací jazyk C.* Brno : Computer Press, a.s., 2006. 286 s. ISBN 80-251-0897-X.

[8] KORANNE, Sandeep. *Handbook Of Open Source Tools.* New York : Springer, 2010. 484 s. ISBN 144197718X.

[9] LAMOŠ, František; POTOCKÝ, Rastislav. *Pravdepodobnosť a matematická štatistika : štatistické analýzy.* Bratislava : Alfa, 1989. 344 s. ISBN 8005001150.

[10] MARTINEK, David. *Jak na projekty v jazyce C* [online]. 11.2.2009 [cit. 2011-05-21]. Profilování a optimalizace programů. Dostupné z WWW: <http://www.fit.vutbr.cz/~martinek/clang/profiling.html>.

[11] PSUTKA, Josef, et al. *Mluvíme s počítačem česky.* Academia, 2006. 746 s. ISBN 8020013091.

[12] RODMAN, Robert D. *Computer speech technology*. Nordwood, MA : Artech House Inc., 1999. 344 s. ISBN 0890062978.

[13] SMÉKAL, Zdeněk. *Číslicové zpracování řeči*. Ústav telekomunikací FEKT VUT v Brně, 2009. 123 s.

[14] SOHN, Jongseo; SUNG, Wonyong. A VOICE ACTIVITY DETECTOR EMPLOYING SOFT DECISION BASED NOISE SPECTRUM ADAPTATION. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference*. Seattle, WA, 1998. s. 365 - 368 vol.1. ISBN 0-7803-4428-6 , ISSN 1520-6149.

[15] SOHN, Jongseo; KIM, Nam Soo; SUNG, Wonyong. A Statistical Model-Based Voice Activity Detection. In *Signal Processing Letters, IEEE*. Leden 1999. s. 1-2. ISSN 1070-9908.

[16] SOVKA, Pavel; POLLÁK, Petr. The study of speech/pause detectors for speech enhancement methods In *In Proceedings of 4th European Conference on Speech Communication and Technology* [online]. Madrid, Spain. 1995 [cit. 2010-11-23]. Dostupné z WWW: <http://noel.feld.cvut.cz/speechlab/publications/003_eurospeech95.pdf>.

[17] STEJSKAL, Vojtěch. *Automatická segmentace řeči a identifikace pauz*. Brno, 2006. 100 s. Dizertační práce. VUT v Brně.

[18] SYSEL, Petr. Statistical Based Voice Activity Detector. *In Proceedings of International Conference on Signals and Electronic Systems.* Lodz, Poland: Lodz, 2006. s. 687. ISBN: 83-921172-5-5.

[19] TANYER, S.Gökhun; ÖZER, Hamza. Voice activity detection in nonstationary noise. In *Speech and Audio Processing, IEEE Transactions*. Vol.8, No. 4., Červenec 2006. s. 478 - 482 . ISSN 1063-6676.

[20] VONDRÁŠEK, Martin; POLLÁK, Petr. Methods for Speech SNR estimation: Evaluation Tool and Analysis of VAD Dependency. In *Radioengineering April 2005, Volume 14, Number 1* [online]. Brno, Duben 2005 [cit. 2010-12-06]. Dostupné z WWW: <http://www.radioeng.cz/fulltexts/2005/05_01_06_11.pdf>. ISSN 1210-2512.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BLAS Basic Linear Algebra Subprograms

CPU Central Processing Unit

DFŘ Diskrétní Fourierova řada

DFT Diskrétní Fourierova transformace

FFT Rychlá Fourierova transformace

GNU GNU's Not Unix

GSL GNU Scientific Library

ROC Receiver operating characteristics

SNR Signal to Noise Ratio - Poměr odstupu signálu od hluku

VAD Voice Activity Detector - Detektor hlasové aktivity

SEZNAM PŘÍLOH

A OBSAH ELEKTRONICKÉ PŘÍLOHY

Přiložené CD obsahuje elektronickou verzi diplomové práce ve formátu PDF a zdrojové kódy aplikace.

Adresářová struktura přílohy diplomové práce na CD:

`/C/system1/`	-	soubory se zdrojovým kódem systému pro detekci v jazyce C spolu s projektovými soubory vývojového prostředí Eclipse
`/matlab/`	-	zdrojové kódy systému detekce v prostředí Matlab
`/nahravky/`	-	databáze nahrávek vozidel a mluvené řeči
`/octave/`	-	zdrojové kódy pro detekci v prostředí Octave